QALoad 5.5 Help

Using the Player, Using the Script Development Workbench, and Using Analyze

Technical support is available from our Customer Support Hotline or via our FrontLine Support Web site.

Customer Support Hotline:
1-800-538-7822

FrontLine Support Web Site:
http://frontline.compuware.com

This document and the product referenced in it are subject to the following legends:

Table Of Contents

Player

Overview of the QALoad Player

The QALoad Player simulates one or more virtual-users running C-based scripts. These scripts mimic user activities to load test the application, network, and server components of a client-server system.

The QALoad Player is used to simulate multiple clients sending middleware calls back to a server. Generally, these are database SQL calls — although other types of middleware layers can also be tested. When running virtual user simulation, QALoad Player can emulate multiple users from a single platform using the multi-tasking features of 32-bit Windows. The number of users that a single hardware system can emulate is determined by the processor speed, main memory size, middleware layer, and simulated transaction rate. Please contact your QALoad distributor for further sizing information.

Once started, QALoad Player is designed to function entirely in the background without any direct user interaction. All commands to QALoad Player come from the QALoad Conductor. In fact, once QALoad Player has been started, the only interaction you may have with it is to change startup parameters or to save the contents of the display window to a file.

Citrix and SAP 6.20/6.40 scripts play back in a virtual user window on the desktop. For SAP, you can enable or disable the VU window from the Conductor's Custom middleware options dialog box. Citrix replay sessions are minimized by default, but can be restored on the desktop.

QALoad Player menus

The following menus are available from the QALoad Player:

File menu

Edit menu

View menu

Options menu

Help menu

Installing UNIX Players

For information about installing UNIX Players, please refer to the QACenter Performance Edition Installation and Configuration Guide.

You can access this guide by clicking Start>Programs>Compuware>QALoad>Documentation>Installation and Configuration Guide.

Tuning QALoad Player for use with Oracle

Oracle version 7 SQL*NET puts significant demands on the system running QALoad Player by demanding at least 1MB of physical memory and approximately 3MB of virtual memory per simulated user. Compuware recommends you follow these guidelines when using Oracle to optimize QALoad Player performance:

- Set the Executing Threads Startup Interval parameter on the Player Configuration dialog box's Startup Parameters tab to between 2,000 and 4,000 milliseconds.

- Unless your application continually logs in and out of Oracle, move the logon commands (DO_olog and its associated DO_ologof) outside the Begin_Transaction/End_Transaction loop, where the Oracnvrt program places them by default.

Dialog Box and Field Description

QALoad Player Main Window

The QALoad Player Main Window is divided into two parts:

- The top portion contains fields, buttons, and options that help you configure the Player for script validation. When an actual load test is in progress, this area displays the following information:
  - Version: The version of the QALoad Player.
  - Player Name: The network name assigned to the Player workstation.
  - Player Address: The network address of the Player workstation.
  - Player Port: The port number on this Player workstation being monitored by the QALoad Conductor.
  - Player is running... the type of virtual users this Player is running.
  - The number of virtual users and transactions this Player is running.
- The bottom portion of the Player Main Window displays Player messages while a script is running.

This section describes the configuration options on the top portion of the Main Window.

Fields and Buttons

Compiled Script

Navigate to the compiled script (.dll) to validate.

Users

Type the number of users to emulate when validating the selected script. Compuware recommends one user for script validation.

Transactions

Type the number of transactions to run when validating the selected script. Compuware recommends one transaction for script validation.

Start

Click the Start button to begin script validation. Player messages will display below.

Abort

Click the Abort button to stop all virtual users immediately.

Exit

Click the Exit button to exit the load test gracefully, when each virtual user is finished.

Debug Data

Select this check box to have the Player display a debug message indicating which command the script is executing and to generate WWW replay log files.

RR__FailedMsg

Select this check box to view, in the Player window, the point where a middleware command within your script fails.

Check Points

Select this check box if you want to display the Check Point command response times in the Player window.

Auto Clear

Select this check box to automatically clear any messages from the bottom portion of the window before running a new script.

Abort on Error

Select this check box to abort script execution when an error is encountered.

Create Timing File

Select this check box to create and save a Player timing file for this Player to the default QALoad timing file directory (normally `\Program Files\Compuware\QALoad\TimingFiles`).

Run As

Select if this Player should run scripts as thread- or process-based.

Save As

Use this dialog box to save a text file of the messages reported by Player during a test, or to save an existing buffer with a different name.

Access this dialog box from the File menu by selecting Save Buffer or Save Buffer As.

Player configuration

Use this dialog box to set startup parameters for Player. The default startup parameters are saved in the player section of the QALOAD.INI file.

Access this dialog box from the Options menu by selecting Player Configuration.

Runtime tab

Player Name: This is the name that the Player will report to the QALoad Conductor during a request. It may be any string of alphanumeric characters, provided that the length does not exceed 10 characters and there are no embedded spaces.

Compiled Scripts: This field points to the directory which will hold the compiled scripts. When a test is started, Player looks for scripts in this directory. The configuration screen will verify that the directory exists.

Compuware recommends that you use a directory on a networked drive to hold the compiled scripts. Otherwise you will need to manually copy the script files to each Player system whenever a script changes.

Local Datapool: This field points to the directory which will hold the local datapool file referenced by this Player workstation.

Timing File: This field points to the default directory where the timing files are located.

Java tab

jvm.dll directory: (optional) This is the directory where the JVM.DLL file is located. If specified, this JVM.DLL will be used to run the Java scripts from a standalone Player; otherwise, the entry specified in the Compiler Settings tab of the Configure QALoad Script Development Workbench dialog box will be used.

How to ...

Installing UNIX Players

For information about installing UNIX Players, please refer to the QACenter Performance Edition Installation and Configuration Guide.

You can access this guide by clicking Start>Programs>Compuware>QALoad>Documentation>Installation and Configuration Guide.

Transferring scripts to a UNIX Player

Normally, the appropriate script is automatically uploaded from the QALoad Conductor to the Players and compiled at runtime. However, if it is ever necessary to manually transfer a script, use the procedure that follows.

Note: The machine where the QALoad Script Development Workbench is installed must have Winsock-based TCP/IP to transfer a script to the UNIX machine where you wish to run it.

To transfer a script:

The following procedure describes how to transfer a script file from the Windows workstation where the QALoad Script Development Workbench resides to the system running the QALoad Player.

1. Access the Script Development Workbench.
2. From the **Session** menu, choose the middleware session you want to start.
3. In the **Workspace Pane**, click the **Scripts** tab.
4. On the **Scripts** tab, select the script you want to transfer.
5. From the **Tools** menu, choose **FTP** to open the FTP Transfer dialog box. Note that the file name you selected to transfer appears in the **File to Transfer** field.
6. Enter the **Host Name**, **User Name**, **Password**, and **Destination Directory**.
7. Click **Transfer** to send the file to the system where your *QA*Load Player is installed.
8. If you want to save the information you have entered for subsequent transfers, click **Save Settings**.
9. Click **Close/Abort** to exit the FTP Transfer dialog box.

Validate a script

To validate a script, follow these steps:

1. In the Compiled Script field, browse for the compiled script DLL you want to validate. Compiled scripts are usually located in the directory `\Program Files\Compuware\QALoad\Scripts`.
2. Type a value in the Number of Users field. Compuware recommends one user for script validation.
3. Type a value in the Transactions field. Compuware recommends one transaction for script validation.
4. Select any appropriate options to the right of the Compiled Script field. These options determine the type and amount of data that will display in the Player Main Window. For descriptions of each options, see the topic QALoad Player Main Window .
5. In the Run As area, select whether the transaction should run as thread- or process-based.
6. Click Start to run the script. The Player Main Window will show the script's progress. If the script runs successfully, it is valid to use in a load test.

Script Development Workbench

Overview of the Script Development Workbench

The QALoad Script Development Workbench is the QALoad component used to develop load test scripts. It contains the facilities you need for recording transactions such as function calls or request/response interactions placed by your Windows application. The recorded transaction, called a capture file, contains raw data that must be converted to an editable test script based on C, C++, or Java, depending upon which middleware environment is under test.

After converting the recorded transaction to a script, you can use the Script Development Workbench's script editor and other functionality to make any necessary modifications to your script. For example, maybe you had to sign on to a Web server with a user name and password as part of your recorded transaction. At test time, when multiple virtual users are running your test script, you might want each user to have a different user name/password combination. You can use the Script Development Workbench to create a re-usable pool of user name/password combinations, saved as a datapool file, and edit your script to extract values from that file at test time. QALoad provides scripting commands for situations like that, and provides a Function Wizard and online language reference, both available right from the editor, to help you locate and insert the right commands.

When you are satisfied with your test script, you can compile it directly from the Script Development Workbench. And, finally, add it to a load test in the QALoad Conductor.

In short, to produce a usable test script you:

1. **Record** a transaction into a capture file (.cap).
2. **Convert** the capture file to an editable script.
3. **Edit** the script.
4. **Compile** the script.

Setting a default middleware session

To set a particular middleware as the default for new sessions:

1. Access the *QA*Load Script Development Workbench.
2. From the **Session** menu, choose the name of the middleware session you want to open. The **Default Session Prompt** opens. The Default Session Prompt didn't open?
3. Select the **Make this my default Session** check box .
4. Click **OK**.

Configuring the Script Development Workbench

The first time you use the QALoad Script Development Workbench, you should set options to determine a working directory QALoad can use for temporary files, compiler settings, and other general options related to the behavior of the QALoad Script Development Workbench.

To set a working directory:

1. Access the Script Development Workbench.
2. From the **Session** menu, choose the session you want to start.
3. From the **Options** menu, choose **Workbench**.
4. Set any appropriate options. For a description of the available options, press **F1** from the Configure Script Development Workbench dialog box.

    Note: Compuware recommends that you always select the Automatically Compile Scripts and Automatically Convert Capture options.

5. Click **OK** to save your settings.

Developing, Converting, and Compiling a Test Script

When you are satisfied with your test script, you can compile it directly from the Script Development Workbench. and add it to a load test in the QALoad Conductor.

To access the Script Development Workbench and develop a test script:

1. Click **Start>Programs\Compuware\QALoad\Script Development Workbench**.
2. From the Session menu, select the appropriate middleware.
3. Choose the middleware name from the Session menu or by clicking the appropriate button on the toolbar. The Default Session Prompt opens.
4. Click OK.

Note: If this middleware type should be the default every time you open the Script Development Workbench, select the check box Make this my default session. If you do not want to be prompted to set a default middleware, clear the Enable default session checking check box. You can also turn default session checking on or off from the Configure Script Development Workbench dialog box at any time.

To set up automatic conversion and compilation:

1. From the **Script Development Workbench** menu, choose **Options>Workbench**.
2. On the Configure Script Development Workbench dialog box, in the **Record Options** area, select the check box **Automatically Convert Capture.**
3. Click the **Compiler Settings** tab.
4. Select the check box **Automatically compile scripts**.
5. Select the check box **Prompt before overwriting script** to ensure that a script isn't overwritten accidentally.
6. Click **OK** to save your settings.

The Script Development Workbench automatically converts a capture file when you stop the recording process and compile the resulting script. You are prompted if a script by the same name already exists, so that you can decide whether to overwrite an existing script or to save your script under a different name.

Sessions

EasyScript Sessions

When you first open the Script Development Workbench, you can set general options related to which panes to display, your compiler, and so on, but you can't begin any middleware-specific activities, such as recording a transaction, until you open an EasyScript Session. Opening an EasyScript Session tailors the Script Development Workbench to a specific middleware environment, providing you with all the appropriate options and functions for your scripting needs.

To open an EasyScript Session, choose your middleware type from the Session menu, or click the appropriate toolbar button. Once a session is open, the Workbench interface will change.

You can also open a Universal session to record calls from multiple middlewares within a single session.

Using the Universal session

The Universal session allows you to record calls from multiple middleware applications within a single Script Development Workbench session. You might use the Universal session in cases where your application accesses an additional application that uses a different protocol.

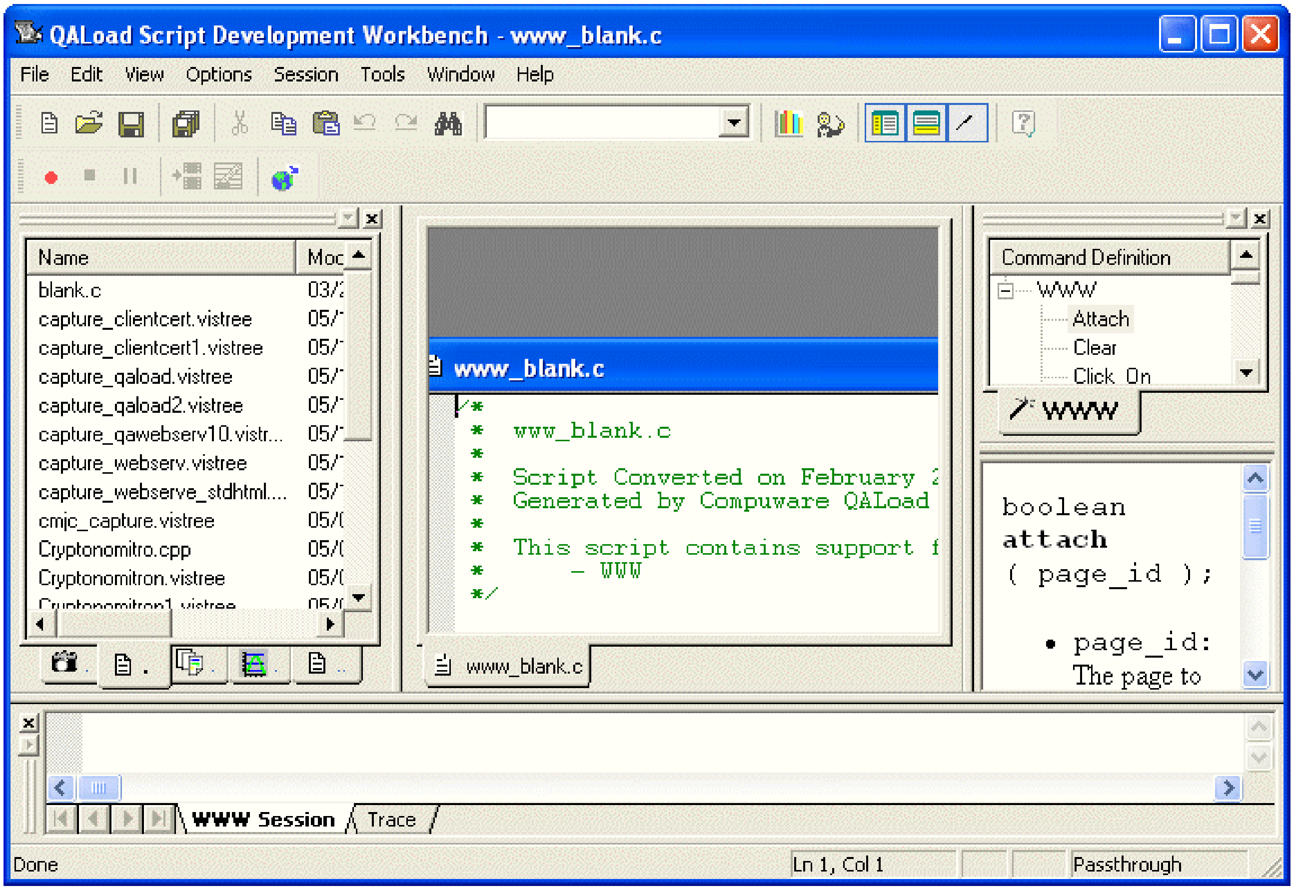
For example, your browser might download and open a Java applet which then communicates with a Winsock server. If you recorded that activity using a simple WWW session, the Script Development Workbench would only record the HTTP requests that downloaded and opened the Java applet. Recording that transaction with the Universal session ensures that you record the HTTP requests from the browser as well as the Winsock-based communication between the Java applet and the Winsock server — all within a single script.

You start and record from a Universal session exactly like a single middleware session with one difference — after starting a Universal session you must select which middleware applications to record.

The Script Development Workbench main window

The QALoad Script Development Workbench main window is divided into dynamic panes that you can hide or show as needed by selecting commands from the View menu.

Hint: Click on a pane in the following graphic for a description of that pane. Use your scroll bars to see the rest of the graphic.



Menus and toolbar buttons

The QALoad Script Development Workbench menus and buttons change depending on whether you have an EasyScript Session open.

Menus and toolbars without an open EasyScript session

The following menus and toolbars are available when an EasyScript Session is not open.

File
View
Options
Session
Tools
Help
Toolbar Buttons

Menus and toolbars with an open EasyScript session

The following menus and toolbars are available when an EasyScript Session is open.

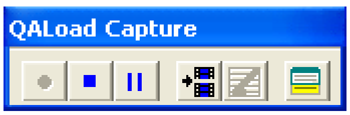
File
Edit
View
Options

Session
Tools
Window
Help
Toolbar Buttons
Recording toolbar

Recording toolbar

The Recording toolbar is a floating toolbar that is launched automatically when you start recording a transaction.

Click each button on the following toolbar image for a description of its functionality.



Developing a Test Script

Recording a Transaction

Recording middleware calls

QALoad begins recording before starting your application, ensuring that any early startup activity is recorded.

Hint: You can save yourself some steps later by setting options now to automatically convert your recorded capture file and compile it into a script.

To record a middleware call:

1. Open an appropriate middleware session in the *QA*Load Script Development Workbench.
2. (Oracle Forms Server only) Choose **Options>Workbench**, then click the **Compiler Settings** tab. In the **Java** section of the dialog box, set the location of your Java files for recording.
3. Select **Session>Record>Start**. *QA*Load launches your application and any proxies, if necessary, and begins recording any calls.
4. Run the desired user operations using your application.
5. (WWW only) If you are capturing SSL requests using EasyScript for Secure WWW, the browser generates one or more prompts indicating the following:

   ! It does not recognize the authority who signed the server certificate.

   ! The server certificate presented by the Web site does not contain the correct site name.

   When you receive these prompts, click the browser’s Next or Continue button so you can connect to and exchange information with the desired Web site.

6. (Optional) At any time during the recording process, you can insert any necessary commands or comments into the capture file.

7. (Optional) At any time during the recording process, you can insert any necessary commands or comments into the capture file.
8. When you have recorded a complete transaction, stop the application from which you are recording.
9. When you finish, click **Stop Record.** You are prompted to save your capture file. By default, capture files (.cap) are saved in the `QALoad\Middlewares\<middleware_name>\captures` directory.

Note: If QALoad is not able to record from your application, try QALoad's alternate procedure for recording.

Inserting commands/comments into a capture file

You can insert commands or comments while recording a capture file.

To insert commands/comments into a capture file:

1. On the Recording toolbar, click **Insert Command**. The toolbar expands into a window where you can select options for inserting commands into your capture file.
2. In the **Command Type** area, select whether you want to insert a comment or a begin/end checkpoint.
3. In the **Command Info** area, type your comment or a description of the checkpoint.
4. Do one of the following:
   - Click Insert to insert your comment or checkpoint command into your capture file.
   - Click Insert Command again to close the expanded window without inserting a command.
5. Continue recording your transaction as usual.

Converting a Transaction to a Script

Converting

A capture file contains all the raw data that was recorded, but it needs to be converted into an editable script file before you can proceed. The script file can then be open in the Script Development Workbench editor and edited as needed.

To convert a capture file to a script:

1. Access the *QA*Load Script Development Workbench. Details.
2. From the **Session** menu, choose the session you want to start.
3. If you have not already done so, set conversion options.
4. In the **Workspace Pane**, click the **Captures** tab.
5. Click the capture file you want to convert and click **Options>Convert**. The capture file is converted to a script. In the Workspace Pane, click the Scripts tab to view the list of scripts you have converted.

Note: If an Error/Warning Summary opens in the Output Pane, resolve any errors.

6. Compile the script.

Note: You can set an option to automatically convert your recorded transactions into scripts. How?

Compiling a test script

A QALoad script is a real C-, C++-, or Java-based script, and therefore needs to be compiled before it can be used. QALoad works with your existing compiler to compile usable test scripts. If you make changes to an existing script, you must re-compile it before you can successfully use it in a test. If you add an uncompiled or out-of-date script to a load test, the QALoad Conductor will prompt you to compile the script.

Customizing a Script

Using the Function Wizard

The Function Wizard allows you to quickly and easily edit your script by choosing from the QALoad commands available to your script and inserting them with a click of your mouse.

The Function Wizard is located in the Script Development Workbench in a pane on the right side of the window. You can enable or disable the Function Wizard from the View menu or by clicking the Show or Hide Function Wizard button on the toolbar.

The Function Wizard lists all functions that are valid to use in your open script. Functions are grouped in logical sections within the top window of the wizard. When you highlight a function in the top window of the wizard, the lower window will list a description of that function and its parameters.

To insert a function into your script, locate it in the Function Wizard and then simply drag-and-drop it into your script.

The function will be written into your script at the point you chose. When you insert a function using the wizard, a text box opens showing the proper syntax and parameter options. (The text box may not appear if an associated variable or object has not been declared in the script.) As you edit the function's parameters, the text box highlights the parameter that is currently being edited.

Note for ADO scripts: After inserting an ADO method, change the # sign to the appropriate object number.

Using custom counters and messages

QALoad allows you to define your own counters and insert messages into your script, where they are written to your timing file and are viewable in Analyze or at runtime in the Conductor.

Counters can be either cumulative or instance. This determines how they are graphed in Analyze:

- For a cumulative counter, Analyze keeps a running sum of the counter while graphing verses elapsed time. This type of counter is used for all the WWW error counters. Each time a WWW error occurs, a value of 1 is written for that counter. When looking at a detailed view in Analyze, you can see at what times that error occurred. When you graph a counter in Analyze, the graph will show the total number of occurrences verses the elapsed time.
- For an instance counter, Analyze graphs each value directly. No summing of previous values is done.

Counters must be added manually using the QALoad commands DEFINE_COUNTER and COUNTER_VALUE. Messages must be added manually using the QALoad command SCRIPT_MESSAGE.

The following sample script illustrates both script counters and messages:

```
#include <stdio.h>
#include "smacro.h"
#include "do_www.h"

int rhobot_script( PLAYER_INFO *s_info )
{
char  buf1[256];
int   id1, id2, id3, id4;

DEFINE_TRANS_TYPE( "ScriptCounters " );
DO_InitHttp(s_info);

// "Counter Group", "Counter Name", "Counter Units (Optional)",
// Data Type, Counter Type.

id1 = DEFINE_COUNTER( "Cumulative Group", "Cumulative long",
                      0, DATA_LONG, COUNTER_CUMULATIVE);
id2 = DEFINE_COUNTER( "Cumulative Group", "Cumulative float",
                      0, DATA_FLOAT, COUNTER_CUMULATIVE);
id3 = DEFINE_COUNTER( "Instance Group", "Instance long",
                      0, DATA_LONG, COUNTER_INSTANCE);
id4 = DEFINE_COUNTER( "Instance Group", "Instance float",
                      0, DATA_FLOAT, COUNTER_INSTANCE);

SYNCHRONIZE();
BEGIN_TRANSACTION();

// add value to cumulative counter 1
COUNTER_VALUE( id1, 1 );
DO_SLEEP( 2 );

// add value to cumulative counter 2
COUNTER_VALUE( id2, 1.5 );
RND_DELAY( 6 );

// add value to instance counter 1
COUNTER_VALUE( id3, s_info->nRndDelay );

// add custom message for this user
wsprintf( buf1, "User %d slept for %d milliseconds during transaction %d",
          s_info->nAbsVUNum, s_info->nRndDelay, s_info->s_trans_count );
SCRIPT_MESSAGE( "User Messages", buf1 );
DO_SLEEP( 2 );

// add value to instance counter 2
// relative user number plus pi times the current transaction number
COUNTER_VALUE( id4, s_info->nRelVUNum + ( 3.14159 * s_info->s_trans_count ) );

END_TRANSACTION();
DO_FreeHttp();
REPORT(SUCCESS);
EXIT();

}
```

Defining checkpoints

Checkpoint statements collect timings of events, such as the execution of SQL statements. If you manually insert checkpoint statements into your capture file during the recording process, or if you select the Include Default Checkpoint Statements conversion option before converting a script, your script includes checkpoints.

Otherwise, you must manually insert checkpoints in your scripts to collect timings.

Defining transaction loops

If you did not insert begin-and end-transaction commands into your capture file, QALoad's Convert facility automatically places begin-and end-transaction commands at the start and end of the recorded sequence. QALoad scripts execute the code between the begin-and end-transaction commands in a loop according to the number of times you specify in the QALoad Conductor when setting up a test.

Depending on how you completed your recording, you may want to move one or both of these transaction commands to another place in the script to more accurately define the transaction that runs during the load test.

For example, let's say during the recording process you log in and log out as part of the procedure. However, during the load test you do not want to log in and log out as part of every transaction. To avoid a login and logout with every procedure, move the begin- and end-transaction commands so the login and logout commands are outside of the transaction loop.

Simulating user-entered data

When you create a script, you probably have some constant data embedded in the script, for example, an employee number, that automatically enters your application's input fields while recording. If you run a load test using this script, the script uses the same data for each transaction. To run a realistic test, you can modify the script to use variable data from a datapool file. By varying the data input over the course of a test, the behavior more realistically emulates the behavior of multiple users. You can use the QALoad Script Development Workbench to create, maintain, and use datapool files (.dat) to insert variable data into your scripts.

A datapool can be defined as either central or local:

- A central datapool is a datapool that resides on the same workstation as the *QA*Load Conductor, and is available to any Player system on the network that requests it from the *QA*Load Conductor. A central datapool is controlled by the *QA*Load Conductor, and you use the *QA*Load Conductor to set any options relating to a central datapool.
- A local datapool is a datapool that resides on a Player workstation, and is only available to that Player. Because a local datapool resides locally and is only available to the local Player, it does not generate any network traffic. You use the *QA*Load Script Development Workbench to insert local datapools into a script.

The following sections describe how to create and use central and local datapools.

Creating a datapool file

You can create a datapool file using the Script Development Workbench.

To create a datapool file:

1. Open a middleware session in the *QA*Load Script Development Workbench.
2. From the **File** menu, choose **New**.
3. On the **New** dialog box that opens, select **New** from the **Datapools** tree item.
4. In the **Filename** field, type a unique name for your datapool file.
5. In the **Rows:** and **Cols:** fields, type the number of rows and columns your new datapool should have.
6. Click **OK**.
7. Enter your datapool records in the grid that opens in the Workbook Pane.
8. When you are finished entering datapool records, click **File>Save As** to name your datapool file.
9. Click **OK** to save the file. *QA*Load saves the file in your `\QALoad\Datapools` directory.

Modifying a datapool file

You can modify a datapool file using the Script Development Workbench.

To modify a datapool file:

1. In the Workspace Pane, click the **Datapools** tab.
2. Double-click the datapool file you want to modify. The datapool file opens in the Workbook pane.
3. Make the appropriate changes and save the file.

Using a central datapool file in a script

You assign a central datapool file to a specific script by selecting the datapool file and setting any appropriate options using the Conductor. Each script can use a single central datapool. The central datapool is available to all Player workstations running the test. The following procedures describe how to assign and extract data from a central datapool. These procedures assume you have already created the datapool file as described above.

Assigning a central datapool file

1. With a session ID file open in the *QA*Load Conductor, click the **Script Assignment** tab.
2. In the **External Data** column for the selected script, click the **Browse** button.
3. In the **External Data** dialog box, navigate to the datapool you wish to use. Select it and click **Open**.
4. If you wish to re-use the datapool records when the script reaches the end of the file, select **Rewind**. To only use each record once, and then discard it, select **Strip**.
5. When you are done, click **OK**.

Using data records from a central datapool file

To use data from a central datapool in your load test, you will have to modify your script. Typically, you will read one record per transaction.

To add datapool statements to your script:

1. With your script open in the *QA*Load Script Development Workbench, navigate to the place where you want to insert a datapool variable and highlight the text to replace.
2. From the **Session** menu, choose **Insert>Datapool**. The **Insert New Datapool** dialog box appears.
3. Select a datapool from the list and click **OK**, or click the **Add** button to open the **Select Datapool** dialog box where you can choose a datapool file to add to your test.
4. When you are finished, the *QA*Load Script Development Workbench places several datapool functions into your script, denoting them with markers so you can easily identify them.

Using local datapool files in a script

You assign a local datapool file to a specific script by selecting the datapool file and setting any appropriate options using the QALoad Script Development Workbench. Each script can use up to 64 local datapools. Use the following procedures to assign and extract data from a local datapool file. These procedures assume you have created a datapool as described above.

Assigning a Local Datapool

1. Open a session in the *QA*Load Script Development Workbench.
2. In the Workspace pane, click the **Scripts** tab.
3. On the **Scripts** tab, double-click on the appropriate script name to open it in the Workbook pane.
4. From the **Session** menu, choose **Insert>Datapool**. The **Insert Datapool Commands** dialog box appears.
5. On the **Insert Datapool Commands** dialog box, click the **Add** button. The **Select Datapool** dialog box opens.
6. In the **Type** field, select **Local**. Note that you can also choose to insert a central datapool from this dialog box. If you choose to insert a central datapool from here, the *QA*Load Script Development Workbench places the Conductor command `GET_DATA` into the script just after the `BEGIN_TRANSACTION` command, bookmarks the command in the margin of the script, and uses any options set for the specified datapool in the *QA*Load Conductor.
7. In the **ID** field, give the datapool a unique identifier. The name can contain alphanumeric characters only. Use underscores (_) for spaces. This ID will help you identify the datapool in your script, for example `“ACCOUNT_NUMS”`.
8. In the Filename field, type (or browse for) the fully qualified path of your datapool file. For example: `c:\Program Files\Compuware\QALoad\Workbench\<middleware_name>\Scripts\datapool.dat`
9. If you wish to re-use the datapool records when the script reaches the end of the file, select **Rewind at End of File**. To only use each record once, and then discard it, clear this option.
10. When you are finished, click **OK**. The selected datapool is displayed on the **Insert New Datapool** dialog box.
11. Click **OK**. The *QA*Load Script Development Workbench will place a `#define` statement identifying the datapool file near the beginning of your script, and place the datapool commands `OPEN_DATA_POOL`, `READ_DATA_RECORD`, and `CLOSE_DATA_POOL` at the default locations in the script. These statements will be bookmarked in the margin for easy identification.
12. When you are finished modifying the script, save any changes.

For detailed information about any of these commands, refer to the Language Reference section.

Using Data Records from a Local Datapool File

To use data from a local datapool file you will have to modify your script to read data records and fields at the appropriate place in the script. Datapool files should typically be opened with the statement `OPEN_DATA_POOL` just before the `BEGIN_TRANSACTION` statement, then datapool fields can be called into the script to replace variable strings. The `OPEN_DATA_POOL` statement is automatically inserted into your script when you use the QALoad Script Development Workbench to insert your datapool.

1. Read a record from the datapool file using the following command, which reads a single record from the local datapool file you specify:
   `READ_DATA_RECORD(<LOCAL DATAPOOL ID>);`
2. To access the fields of this record, substitute `GET_DATA_FIELD(ACCOUNT_NUMS, n)` expressions in place of variable strings.
3. After the `END_TRANSACTION` statement, close the local datapool file by using the following statement:
   `CLOSE_DATA_POOL( LOCAL DATAPOOL ID );`

Note that this statement is added automatically if you use the QALoad Script Development Workbench to insert your datapool.

For detailed information about any of these commands, refer to the Language Reference section.

Inserting Variable Data with ActiveData Substitution

The QALoad Script Development Workbench allows you to transform string data from quoted constants or substrings into variables. ActiveData variable substitution lets you identify and right-click on a string to declare the selected string a variable within the QALoad script. This facility also lets you select or edit datapool entries more dynamically, making script development easier and more efficient.

To substitute a datapool value or a variable in place of a selected string in your script:

1. Start the appropriate session in the *QA*Load Script Development Workbench.
2. In the Workspace pane, click the **Scripts** tab.
3. On the **Script** tab, double-click the script you wish to open. The script opens in the Workbook pane.
4. In the script, highlight the string you wish to replace.
5. Right-click anywhere in the highlighted string.

   ! To substitute a value from a datapool:

   - — Click ActiveData>Datapool Substitution in the shortcut menu that opens. The ActiveData Datapool Substitution dialog box opens.
   - In the Datapool(s) area, highlight the datapool to use. The contents of the datapool file display below. If the datapool you want to use is not listed, click the Add button to add it to the list of available datapools.
   - In the Field: ID field, type the field number of the specific value to use from the datapool.
   - When you are finished, click OK. The QALoad Script Development Workbench will place a `#define` statement identifying the datapool file at the beginning of your script. It will also insert the datapool commands `OPEN_DATA_POOL`, `READ_DATA_RECORD`, `GET_DATA_FIELD` and `CLOSE_DATA_POOL` at the default locations in the script, and bookmark them in the margin for easy identification. Refer to the Language Reference section for detailed information about any of those commands.

   ! To substitute a variable:

   - Click ActiveData>Variable Substitution from the shortcut menu that appears. The ActiveData Variable Substitution dialog box opens.
   - Assign a variable name for the selected string in the Variable Name field.
   - Click OK. The QALoad Script Development Workbench will declare the variable at the beginning of your script and substitute the named variable for the selected string. It will also bookmark both locations for easy identification.

6. When you are finished, save your script changes. Compuware recommends that you also compile your script to check for any errors.

Middleware Scripting Techniques

Citrix

Handling Citrix server farms

Citrix servers can be grouped in farms. When load testing, you may want to connect to a Citrix server farm rather than to a specific server. This type of setup load tests the server farm and Citrix load balancing rather than a single server, which provides a more realistic load test.

To record a script that connects to a farm, you must use an ICA file to connect. However, when a capture takes place, a specific server (in the farm) must have a connection. Specify the correct ICA file to connect to the server farm as well as a specific server within that server farm. To verify that your script is connecting to a server farm and not a specific server, assign the server name to one blank space when validating the script. For example:

.
.
.

```
/* Declare Variables */
const char *CitrixServer   = " ";
const char *CitrixUsername = "citrix";
const char *CitrixPassword = "~encr~657E06726F697206";
const char *CitrixDomain   = "qacitrix2";
const int CitrixOutputMode = OUTPUT_MODE_NORMAL;

.
.
.

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("Orders.cpp");

CitrixInit(4);

/* Citrix replay settings */

CtxSetConnectTimeout(90);
CtxSetDisconnectTimeout(90);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, Citrix-Domain);
CtxSetICAFile("PRD desktop.ica");
CtxSetEnableCounters(TRUE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
```

Handling dynamic window titles

Some applications create windows whose titles vary depending on the state of the window. For example, Microsoft Word creates a title based on the default document name at the time of the window creation. During replay, this dynamic title can differ from the window title that was recorded, and the window is not recognized. If this occurs, try the following steps to modify the script:

1. **Ensure that the Enable Wildcard Title Match check box is selected in the Citrix conversion options prior to converting the recording.**
   In the Window Verification group of the **Citrix Convert Options** dialog box, ensure that the **Enable Wildcard Title Match** check box is selected. This check box is selected by default. If you are working with a previously-converted script, ensure that a CtxSetEnableWildcardMatching command exists in the script prior to the BEGIN_TRANSACTION command and that the parameter is set to TRUE.

2. **Verify whether there is an issue with dynamic window titles.**
   When a script fails on validation because the run time window title is different than the expected window title from the recording, it is likely that you are dealing with a dynamic title issue that can be handled by this scripting technique. In this case, the script fails on the CtxWaitForWindowCreate call.

3. **Identify a match “pattern” for the dynamic window title.**
   Note the error message that is returned during validation (or replay). The message indicates the expected window title versus the window title from script playback. Examine the differences in the window titles to create a “match pattern” that recognizes the window title, while ignoring other windows. A match pattern can be a simple substring of the window title or a pattern string using wildcard characters such as ? (to match any single character) or * (to match any number of characters). The examples below illustrate the different match patterns.

4. **Insert a CtxSetWindowMatchTitle command prior to the CtxWaitForWindowCreate call for the dynamic window.**
   When adding the SetWindowMatchTitle command, ensure that the first parameter contains the correct window object and the second parameter contains the match string in double-quotes.

5. **Validate the script to ensure the CtxWaitForWindowCreate command recognizes the dynamic window name.**
   Run the revised script through validation to ensure that the script succeeds. If the script does not validate successfully, go to step 3 to determine if the match pattern is correct.

Example 1: Using a substring match

In this example, the Microsoft Word application generates a dynamic title when the script is replayed. The dynamic name is a concatenation of the default document that Word creates at application startup with the name of the application. The script is altered to reflect the fact that the string “Microsoft Word” is always part of the window title:

```
// Window CWI_13 ("Microsoft Word") created
CtxSetWindowMatchTitle( CWI_13, “Microsoft Word” );
CtxWaitForWindowCreate(CWI_13);
```

Example 2: Using a wildcard match with the * character

In this example, the SampleClientApp application generates a dynamic title when the script is replayed. The dynamic name is the name of the application followed by the name of the user, beginning with the word “User”. The asterisk (*) wildcard is substituted for a given username, reflecting the pattern of “SampleClientApp – User:” as part of the window title followed by an arbitrary user name:

```
// Window CWI_13 ("SampleClientApp – User: John") created
CtxSetWindowMatchTitle(CWI_13,“SampleClientApp – User: *” );
CtxWaitForWindowCreate(CWI_13);
```

Example 3: Using a wildcard match with the ? character

In this example, the RandomValue application generates a dynamic title when the script is replayed. The dynamic name is the application followed by a random single digit. The question mark character is substituted for the single digit to reflect the pattern that begins “RandomValue: ”, followed by single digit:

```
// Window CWI_13 ("RandomValue: 0") created
CtxSetWindowMatchTitle( CWI_13, “Sample Application: ?” );
CtxWaitForWindowCreate(CWI_13);
```

Handling dynamic windows

During conversion, CtxWaitForWindowCreate calls are added to the script for each named window creation event. During replay, some dynamic windows that were in the capture may not appear, which causes the script to fail because a wait point times out. To avoid script failure in this circumstance, comment out the CtxWaitForWindowCreate commands that may be referencing dynamic windows.

Handling dynamic windows that require user interaction

Some windows that require user action before normal script processing can proceed may appear intermittently during replay. One example commonly encountered with Citrix is the ICA Seamless Host Agent window. This window, if it appears, requires user action or the script may fail.

To work around this issue, follow these steps:

1. Capture a session in which the dynamic window appears and the user performs the action to dismiss the window. This may require multiple attempts to capture the window. Once this is captured in a recording, save the script as a temporary script.
2. If the window did not appear in the primary script, extract the code snippet from the temporary script that acts on the dynamic window and insert it into the real script. The code usually consists of a CtxPoint command and a CtxClick command for this window. Insert the commands after the CtxWaitForWindowCreate command for the dynamic window. In addition, extract and insert the Citrix window information object constructor call and delete call to the relevant parts of the script, changing the object name to avoid conflicting with existing window objects. Ensure that the additional code is not inserted between a CtxPoint command and a CtxClick command in the primary script.
3. Add a special CtxSetWindowMatchTitle command immediately before the CtxWaitForWindowCreate command. The first parameter of the CtxSetWindowMatchTitle command should be the correct window object. The second

parameter contains a special wildcard match "*" that enables the CtxClick command to accept any window title, which ensures that even if the matching window does not appear, the command still executes successfully.

4. If the window appears in the primary script, comment out the CtxWaitForWindowCreate command for the dynamic window. Because the window itself may not appear, the CtxWaitForWindowCreate command should be commented out.
5. Validate the script. If the validation is not successful, ensure that steps 2-4 were performed correctly.

In the following example's scenario, the ICA Seamless Window Agent window does not appear in the primary script, but appears intermittently when the primary script is replayed, causing those replay sessions to fail. A second Citrix script, which includes the window appearance, is recorded and the CtxPoint and CtxClick commands are extracted from this script and inserted into the primary script, with the window object changed to match the object in the primary script. In addition, the Citrix window object constructor call and delete call are added in the appropriate places in the script, and the CtxClick command is changed to refer to this object. In the following example, the text in bold represents code that was manually inserted into the location in the primary script where the window appears in the secondary script.

```
CtxWI *CWI_99 = new CtxWI(0x10052, "ICA Seamless Host Agent", 0, 0, 391, 224);
...
CtxSetWindowMatchTitle( CWI_99, "*" );
CtxPoint(190, 203);
CtxClick(CWI_99, 0, L_BUTTON, NONE);
CtxPoint(300, 400);
...
delete CWI_99; // "ICA Seamless Host Agent"
```

Handling unexpected events in Citrix

The CtxWindowEventExists and CtxScreenEventExists commands can be used to handle unexpected window and screen events in Citrix scripts. When there is a possibility of unexpected dialogs appearing or unexpected screen events occurring, you must modify the script to respond to the changes and continue the load test.

For example, if a script opens a Microsoft Word document that resides on a network, and that document is already open by another network user, an unexpected dialog box appears that prompts the user to choose between continuing to open the document in read-only mode or to cancel it. To prevent script failure, modifications can be made in the script to handle the dialog boxes that appear in this situation.

Generally, to handle unexpected events, you record two scripts. The first script contains a recording of the expected events. The second script should include the unexpected events. Using the CtxWindowEventExists and CtxScreenEventExists functions, create a conditional block of code that handles the dialogs that may appear.

Example

The following script example shows the additional script lines that were added to handle a Word document that is already open by another user on a network. The added lines appear in boldface type.

```
/*
 *  capSave11111-2.cpp
 *
 *  Script Converted on June 21, 2004 at 01:04:17 PM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 50
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
```

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 132 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : No
 * Citrix
 *  General Options                 :
 *   Window Verification            : Yes
 *   Session Timeouts               : Yes
 *    Connect Timeout (s)           : 60
 *    Disconnect Timeout (s)        : 60
 *    Window Creation Timeout (s)   : 30
 *    Ping Timeout (s)              : 20
 *    Wait Point Timeout (s)        : 30
 *   Include Wait Points            : Yes
 *   Enable Counters                : No
 *   Include Unnamed Windows        : Yes
 *   Output Mode                    : Normal
 *  Input Options                   :
 *   Combine Keyboard Input         : Yes
 *   Combine Mouse Input            : Yes
 */

#define CITRIX_CLIENT_VERSION "8.00.60000"
#define CITRIX_ICO_VERSION    "2.4"
#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
            /* Declare Variables */
            const char *CitrixServer      = "qaccitrix";
            const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

            /* Citrix Window Information Objects */
            CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
            CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
            CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
            CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198,
397, 127);
            CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198,
397, 127);
            CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
            CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
            CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
            CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
            CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
            CtxWI *CWI_11 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
            CtxWI *CWI_12 = new CtxWI(0x2006a, "", 200, 186, 156, 287);
            CtxWI *CWI_13 = new CtxWI(0x10138, "", 112, 116, 416, 248);
            CtxWI *CWI_14 = new CtxWI(0x50036, "Microsoft Word", -4, -4, 649, 461);
            CtxWI *CWI_15 = new CtxWI(0x1017e, "Open", 19, 23, 602, 387);
            CtxWI *CWI_16 = new CtxWI(0x20174, "*Microsoft Word", -4, -4, 649, 461);
            CtxWI *CWI_17 = new CtxWI(0x10058, "", 113, 114, 305, 26);
```

```
            CtxWI *CWI_18 = new CtxWI(0x2013e, "Calculator", 66, 66, 261, 253);
            CtxWI *CWI_19 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
            CtxWI *CWI_20 = new CtxWI(0x3006a, "Shut Down Windows", 111, 96, 418, 193);

            CtxWI *CWI_117 = new CtxWI(0x20172, "File In Use", 144, 127, 352, 179);
            CtxWI *CWI_118 = new CtxWI(0x30172, "11111111 (Read-Only) - Microsoft Word", -4,
-4, 649, 461);

            SET_ABORT_FUNCTION(abort_function);

            DEFINE_TRANS_TYPE("capSave11111-2.cpp");

            CitrixInit(1);

            /* Citrix replay settings */
            CtxSetConnectTimeout(60);
            CtxSetDisconnectTimeout(60);
            CtxSetWindowTimeout(30);
            CtxSetPingTimeout(20);
            CtxSetWaitPointTimeout(30);
            CtxSetWindowVerification(TRUE);
            CtxSetEnableCounters(FALSE);
            CtxSetWindowRetries(5, 5000);
            CtxSetEnableWildcardMatching(TRUE);

            SYNCHRONIZE();

            BEGIN_TRANSACTION();

            DO_SetTransactionStart();

            CtxConnect(CitrixServer, CitrixOutputMode);

            // Window CWI_1 ("Warning !!") created 1087837356.454

            CtxWaitForWindowCreate(CWI_1, 2125);

            DO_MSLEEP(1891);
            CtxPoint(246, 267);  //1087837358.797

            DO_MSLEEP(453);
            CtxMouseDown(CWI_1, L_BUTTON, NONE, 246, 267);  // 1087837358.797

            CtxMouseUp(CWI_1, L_BUTTON, NONE, 247, 267); //1087837359.032

            .
            .
            .

            DO_MSLEEP(63);
            // Window CWI_14 ("Microsoft Word") created 1087837397.390

            CtxWaitForWindowCreate(CWI_14, 141);

            DO_MSLEEP(78);
            CWI_14->setTitle("Document1 - Microsoft Word"); //1087837397.468

            // Window CWI_13 ("") destroyed 1087837397.468

            DO_MSLEEP(2468);
            CtxPoint(37, 50);  //1087837400.218

            DO_MSLEEP(282);
            CtxClick(CWI_14, 203, L_BUTTON, NONE); //1087837400.421
```

```
// Window CWI_15 ("Open") created 1087837400.764

CtxWaitForWindowCreate(CWI_15, 344);


DO_MSLEEP(1656);
CtxPoint(132, 99);  //1087837402.671

DO_MSLEEP(250);
CtxDoubleClick(CWI_15); // 1087837402.874

DO_MSLEEP(109);


DO_MSLEEP(1953);
CtxPoint(247, 197);  //1087837404.827

// Window CWI_15 ("Open") destroyed 1087837404.827


if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE,3000,CWI_16))
BeginBlock();
            CtxPoint(337, 265);  //1087837404.905

            // Window CWI_16 ("11111111 - Microsoft Word") created
1087837404.905

            CtxWaitForWindowCreate(CWI_16, 31);

            // Window CWI_14 ("Document1 - Microsoft Word") destroyed
1087837404.905

            DO_MSLEEP(7547);
            CtxPoint(628, 9);  //1087837414.592

            DO_MSLEEP(2141);
            CtxClick(CWI_16, 281, L_BUTTON, NONE); //1087837414.873

            DO_MSLEEP(234);
            // Window CWI_16 ("11111111 - Microsoft Word") destroyed
1087837415.108

            CtxPoint(113, 93);  //1087837418.779

            // Window CWI_17 ("") created 1087837418.779
EndBlock()

///ReadOnly Code Start

else
BeginBlock();

            // Window CWI_117 ("File In Use") created 1087840076.599

            CtxWaitForWindowCreate(CWI_117, 578);

            DO_MSLEEP(2360);
            CtxPoint(358, 283);  //1087840079.068


            DO_MSLEEP(125);
            CtxClick(CWI_117, 281, L_BUTTON, NONE); //1087840079.365


            DO_MSLEEP(109);
```

```
                        // Window CWI_117 ("File In Use") destroyed 1087840079.458

                        // Window CWI_118 ("11111111 (Read-Only) - Microsoft Word") created
1087840079.521

                        CtxWaitForWindowCreate(CWI_118, 63);

                        // Window CWI_115 ("Document1 - Microsoft Word") destroyed
1087840079.521

                        DO_MSLEEP(4766);
                        CtxPoint(631, 3);  //1087840084.490

                        DO_MSLEEP(203);
                        CtxClick(CWI_118, 250, L_BUTTON, NONE); //1087840084.740

                        DO_MSLEEP(93);
                        // Window CWI_118 ("11111111 (Read-Only) - Microsoft Word")
destroyed 1087840084.833

                        DO_MSLEEP(2407);
                        CtxPoint(34, 465);  //1087840087.333

            EndBlock();

///ReadOnly Code End


            DO_MSLEEP(1063);

            DO_MSLEEP(484);
            CtxPoint(112, 93);  //1087837419.654

            DO_MSLEEP(406);
            CtxDoubleClick(CWI_9); // 1087837419.904
            .
            .
            .

            // Window CWI_9 ("Program Manager") destroyed 1087837440.122

            // Window CWI_7 ("") destroyed 1087837440.138

            DO_SetTransactionCleanup();

            CtxDisconnect();


            END_TRANSACTION();

            delete CWI_1; // "Warning !!"
            delete CWI_2; // "Log On to Windows"
            delete CWI_3; // "Please wait..."
            delete CWI_4; // "Citrix License Warning Notice"
            delete CWI_5; // "Citrix License Warning Notice"
            delete CWI_6; // "UsrLogon.Cmd"
            delete CWI_7; // ""
            delete CWI_8; // "ICA Seamless Host Agent"
            delete CWI_9; // "Program Manager"
            delete CWI_10; // ""
            delete CWI_11; // ""
            delete CWI_12; // ""
            delete CWI_13; // ""
            delete CWI_14; // "Microsoft Word"
            delete CWI_15; // "Open"
```

```
            delete CWI_16; // "11111111 - Microsoft Word"
            delete CWI_17; // ""
            delete CWI_18; // "Calculator"
            delete CWI_19; // ""
            delete CWI_20; // "Shut Down Windows"

            delete CWI_117; // "File In Use"
            delete CWI_118; // "11111111 (Read-Only) - Microsoft Word"

            CitrixUninit();

            REPORT(SUCCESS);
            EXIT();
            return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
            RR__printf("Virtual User ABORTED.");

            CitrixUninit();

            EXIT();
}
```

Moving the Citrix connect and disconnect outside the transaction loop

If your load testing requirements for Citrix include creating extended logon sessions, in which the user remains connected to the Citrix server between transactions, review the following tips for recording and script development.

Recording

Perform the following steps during the recording process in order to prepare for moving the connect and disconnect actions outside the transaction loop:

1. Insert a comment such as “Logged in to Citrix” after the Citrix logon but before any windows have been opened.
2. Ensure that all application windows are closed before disconnecting from the Citrix session.
3. Insert a comment such as “Ready to log off Citrix” before the Citrix logoff sequence is initiated. Ensure that the first comment is added after the user has logged on and closed all login-related dialog boxes, but before any applications are started. Similarly, the second comment must be placed after all applications have been closed, but before the user logs off.

Scripting

Comment out the BEGIN_TRANSACTION and END_TRANSACTION calls and add new BEGIN_TRANSACTION and END_TRANSACTION calls at the location where the comments from steps 1 and 3 above were placed. Comment out the calls instead of deleting them so that the original location of these commands can be determined for debugging purposes.

Also comment out the DO_SetTransactionStart and DO_SetTransactionCleanup calls.

Using the CtxWaitForScreenUpdate command

In some situations, a window may vary in how long it takes to refresh on the screen. For example, the Windows Start menu is an unnamed window that can take varying amounts of time to appear, depending on system resource usage. To prevent playback problems in which a mouse click does not synchronize with its intended window, insert the CtxWaitForScreenUpdate command in the script after the action that causes the window to appear. The parameters for the CtxWaitForScreenUpdate command correspond to the X and Y coordinates and the width and height of the window. This command ensures that the window has enough time to display before the mouse click.

OFS

Understanding the C++ script

Oracle Forms Server scripts are produced for Oracle Forms, 6.0, 6i, and 9i (Release 2 and later) recordings. The C++ script executes OFS-related statements by passing the statements in the script DLL to the OFS Java engine that performs the client activities and the client communication with the server. Because the C++ script statements are directly tied to corresponding methods in the OFS Java engine, modifications to the script statements are limited to changing the property parameter values through variablization.

An OFS C++ script contains three main sections: Connection, Application Body, and Disconnect. The QALoad transaction loop includes all three sections by default. The transaction loop can be moved using the guidelines described in Moving the OFS transaction loop. An internal auto checkpoint is created during connection statements and transmission statements.

The C++ script statements are a condensed version of the Java-style script statements. The C++ script statements show the GUI controls in the OFS application and the control properties, which are either control attributes or activities. For example:

```
ofsClickButton( "BUTTON", 52, OFS_ENDMSG, 325 );
```

In this example, the user clicks (property 325) a button (control ID 52). OFS_ENDMSG is a flag that indicates that the GUI activity ends the current OFS Message.

QALoad also allows OFS and WWW statements from a Universal session to be scripted in the C++ script, providing the ability to play back WWW and OFS statements.

Connection statements

The connection script lines in the C++ script vary depending on the type of Forms connection mode that is active. You choose the Forms connection mode on the Oracle Forms Server Recording Options dialog box. Forms connection modes include server-side recording, HTTP, HTTPS, or socket.

Server-side recording is limited to applications that use Forms 9i (applications running in Oracle 9iAS Release 2 and above). HTTP connection mode is available for applications using Forms 9i and for applications using the patched Forms 6i version configured with the HTTP servlet. HTTPS connection mode is strictly for SSL-enabled applications that use Forms 9i. Socket connection mode is for applications that use Forms 6i and lower versions, such as Oracle 11i.

Server-side recording connections

Server-side recording mode contains only one connection statement. The function that is used – ofsSetServletMode – contains the listener servlet value that you entered on the Oracle Forms Server Recording Options dialog box. The first parameter defines the HTTP or HTTPS configuration of the application environment. The second parameter defines the name of the Forms Listener Servlet used by the application. To connect, QALoad internally invokes Oracle's dispatch calls using the two parameters. Oracle's proprietary classes provide the implementation for the HTTP or HTTPS connection. For example:

```
ofsSetServletMode(OFS_HTTP, "http://ntsap45b:7779/forms90/l90servlet" );
```

HTTP connections

HTTP connection mode contains multiple connection statements. To connect, QALoad internally performs Java calls to accomplish the following tasks:

- Define HTTP header properties

- Connect to the Forms Servlet (an HTTP-GET request)
- Set the parameters of the Forms Listener Servlet
- Connect to the Forms Listener Servlet (an HTTP-GET request)
- Set additional HTTP header property for the Listener Servlet
- Connect to the Forms Listener Servlet (an HTTP-POST request). The last connection statement also initiates the required Forms “handshake” and determines the Forms encryption used by the application environment.

For example:

```
ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.9" );
ofsHTTPSetHdrProperty("Host", "ntsap45b:7779" );
ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, "*/*; q=.2"
);
ofsHTTPSetHdrProperty("Connection", "Keep-alive" );
ofsHTTPConnectToFormsServlet(
"http://ntsap45b:7779/forms90/f90servlet?ifcmd=startsession"  );
ofsHTTPSetListenerServletParms( "?ifcmd=getinfo&ifhost=C104444D01&ifip= "192.168.234.1"
);
ofsHTTPConnectToListenerServlet( "http://ntsap45b:7779/forms90/l90servlet");
ofsHTTPSetHdrProperty("Content-type", "application/x-www-form-urlencoded" );
ofsHTTPInitialFormsConnect();
```

HTTPS connections

HTTPS connection mode uses the same connection statements as HTTP mode. To connect, QALoad internally performs the same tasks as the HTTP connection mode plus it performs the SSL connection when the ofsHTTPSDoSSLHandshake function is called. This statement is positioned in the script before the ofsHTTPConnectToFormsServlet function.

Socket connections

Socket mode contains only one connection statement. The function that is used – ofsConnectToSocket – contains the port number and the URL you entered on the OFS Recording Options dialog box to start OFS capture. The port value is the port on which the Forms Server directly listens for Forms traffic. To connect, QALoad uses Java calls to open a Java socket using the parameters, initiate the required Forms "handshake”, and determine the Forms encryption used by the application environment. For example:

```
ofsConnectToSocket("10.10.0.167", 9002 );
```

Application statements

The application statements in the C++ script consist of property statements and transmission statements. Property statements describe the attributes and activities of GUI controls in the application. Transmission statements send the GUI controls and their properties as Forms Message data to the server. There is only one transmission statement: ofsSendRecv. QALoad creates an internal auto checkpoint when this statement is executed. In the following example, the first two (property) statements set the location and size of a FormWindow GUI control. The ofsSendRecv statement sends the GUI control properties to the server.

```
ofsSetWindowLocation( "FORMWINDOW", 6, OFS_ENDMSG, 135, 0, 0);   //Property
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 650, 500);   //Property
ofsSendRecv(1 ); //Transmission
```

Parameters of a property statement:

The parameters of a property statement are arranged in the following sequence:

1. **Captured control name**. If the name is not available, this value is the class name to which the control belongs.
2. **Captured control ID**.

3. **Action type**. This flag indicates if the property is to be added to the current Forms Message or if the property ends the current Forms Message. During playback, each control is treated as a Forms Message. When the current Message ends, *QA*Load translates the control and its properties to binary format. The valid values are:
    - OFS_ADD – add the property to the current Message.
    - OFS_ENDMSG – add the property to the current Message and end the Message.
    - OFS_STARTSUBMSG – add the property of the succeeding nested Message to the current Message.
4. **Property ID**. The Forms version-specific ID of the property.
5. **Property value**. Captured value of the property (optional)
6. **Property value**. Captured value of the property (optional)

For example:

```
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 650, 500);
```

In this example, control ID 6, which belongs to GUI class FORMWINDOW, is resized (PROPERTY 137) to have coordinates 650 and 500. This marks the end of the current Message.

Forms environment statements:

The initial set of statements in the Forms script describes the Forms application environment. In this set, the "version" and the "cmdline" properties are the most important. The version property shows the Forms Builder version used by the application. The version indicates the capabilities of the application. For example, some versions cannot support HTTP connections. The cmdline property shows the Forms configuration parameters passed to the server by the Forms applet. The parameter "record=names" indicates that the application enables GUI control names to be captured. Control names are preferred in multi-threaded playback. The "ICX" parameter indicates that the application uses a Personal Home Page, which requires that you supply OracleAppsLogin information on the Oracle Forms Server Convert options dialog box for the script to run successfully.

In the sample script below, the Forms builder version is 90290 (the version used in Oracle 9iAS Release 2, unpatched). The cmdline property shows "record=forms" which defaults "record=names". The cmdline property does not have the "ICX" ticket parameter.

```
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "90290" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1024, 768);
ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
 "server module=test1.fmx userid=  sso_userid=  debug=no buffer_records=no debug_"
 "messages=no array=no query_only=no quiet=yes render=no host=ntsap45b.prodti.com"
 "puware.com port= record=forms tracegroup=debug log=run1 term=" );
ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "0" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "8421504" );
ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 8, 20);
ofsSetNoRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
ofsSetPropertyString( "RUNFORM", 1, OFS_ENDMSG, 530, "America/New_York" );
ofsSendRecv(1 );
//ClientSeqNo=1|CapTime=1086884188.281|MsgCount=1
```

Sending messages to the server:

The ofsSendRecv statement sends the accumulated GUI controls and their properties to the Forms Server as binary data. This statement represents the point at which the client sends a Forms Terminal Message to the server. In Oracle Forms, the client and the server must end each data block with a Terminal Message before any transmission occurs.

Internally, QALoad varies the binary data transmission depending on the connection mode:

- For server-side recording mode, *QA*Load sends the binary data by invoking Oracle's dispatch calls. Oracle's own classes provide the implementation for the HTTP transmission.
- For HTTP or HTTPS mode, *QA*Load wraps the binary data inside an HTTP stream and invokes Java's HTTP calls.
- For socket mode, *QA*Load sends the binary data directly to the Java socket opened at the connection point.

The ofsSendRecv statement has one parameter: the response code of the captured Terminal Message. The possible values for this parameter are 1 (add), 2 (update), and 3 (close). Typically, when the response code is 3, the Forms Server reacts by removing the GUI controls associated with the client message from the server cache.

A comment line appears after each ofsSendRecv statement that contains script-tracking information. The information on the comment line is also found in the capture file in each ofsSendRecv capture line. The comment line shows the relative sequence of each client request, as represented by a Terminal Message, from the start of the application (e.g. ClientSeqNo=1). The comment line also shows the timing mark of the captured Terminal Message (e.g. CapTime=1086884188.281) and the number of Forms messages contained in the request (e.g. MsgCount=1). The number of Messages can be verified by counting the preceding ENDMSG and STARTSUBMSG flags in the request block. The comment line is useful for debugging playback issues because it readily shows the client request sequence number where the issue is occurring.

Getting the server reply:

During the execution of ofsSendRecv, QALoad also obtains the server's reply and translates the binary Forms data into Forms control values and control properties. The values are also written to the playback log file (in capture file format) if script logging is enabled. The following sample is a server reply:

```
VU 0 :   M|S|2|0|1
VU 0 :   P|S|322|java.lang.Integer|0|151000320
VU 0 :   P|S|279|java.lang.Boolean|0|false
VU 0 :   P|S|525|java.lang.String|AMERICAN_AMERICA.WE8MSWIN1252
VU 0 :   T|S|1|ServerSeqNo=1|MsgCount=76
```

The first line indicates the start of a Forms Message from the server (M|S). The third parameter is an action code (1= add, 2= update, 3= delete, 4= get property value). The fourth parameter is the Class Code of the control (0 = root class). The fifth parameter is the Control ID (1= RunForm).

The second, third and fourth lines are property lines related to the above Forms Message from the server (P|S). The third parameter of each line is the property ID (322). The fourth parameter is the data type of this property (java.lang.Integer). The fifth parameter is the data value. If the value is 0, the data value is in a sixth parameter (false).

The third line is the terminal message line from the server (T|S). The third parameter is the response code associated with the terminal message (1= add, 2= update, = close). The fourth parameter is the relative sequence of the server reply, as represented by a Terminal Message, from the start of the application (e.g. ServerSeqNo= 1). The fifth parameter is the number of Forms messages contained in the reply (e.g. MsgCount = 1). The number of Messages may be verified by counting the preceding M|S flags in the reply block. The fourth and fifth parameters are script-tracking information, which can be useful for debugging a playback issue. If logging is enabled, the log file shows the tracking information, which can make the comparison between server responses and captured responses easier.

Processing large data and delayed response scenarios:

When HTTP or HTTPS connection mode is used, Forms data is wrapped inside the HTTP reply stream. QALoad checks the HTTP header of the reply before processing the Forms data. The HTTP header sometimes indicates that the client needs to perform additional HTTP POST requests to obtain the complete Forms data. This indication occurs when the content-length of the reply is 64000 (a large data scenario), or the content-type is "text/plain" and the HTTP header contains an "iferror: " string (a delayed response/re-post scenario). QALoad performs the necessary POST requests to obtain the complete reply data, and then translates the accumulated reply data to Forms controls and properties.

Disconnect statements

The disconnect script lines vary depending on the Forms connection mode.

- In server-side recording mode, the ofsServerSideDisconnect script statement internally invokes Oracle's dispatch calls to disconnect.
- In HTTP mode, the ofsHTTPDisconnect statement internally makes Java calls to disconnect the main URL connection from the servlet.
- In socket mode, the ofsSocketDisconnect statement closes the socket on which the Forms Server listens for traffic.

Using script logging as a debugging tool

You can debug a playback issue in a C++ script by enabling replay logging. The option for enabling replay logging is located on the Script Assignment tab of the Conductor. For more information about enabling log file generation, see Debugging a script.

In Java-based scripts, logging is not enabled by default. To enable logging, change the parameter of the Logging method to true in the script. For example:

```
oracleForms.Logging( true );
```

When logging is enabled, QALoad writes the client requests and server replies to the playback log file in the same format as the capture file. The playback log file is found in the `\QALoad\LogFiles` directory. When there is an issue during playback, such as the server not responding to a client request, you can compare the capture files and check the differences in the server reply data. Both the capture file and the log file contain tracking information appended to the server's terminal messages. The tracking data contains the relative sequence number of the server reply from the start of the Forms session and the timing mark. The tracking data also shows the number of Forms messages contained in the reply block. The number of messages are based on the number of "M|S" lines prior to the "T|S" lines.

In the following example, the first set of statements shows the logged statements and the second set of statements shows the captured statements. The ServerSeqNo value shows that this is the 8th reply from the server. The MsgCount value of 1 shows that only one Forms Message is included in this reply block.

```
1087419810.000|ofsShowWindow|WINDOW_START_APP|11|OFS_ENDMSG|173|PROPERTY_VISIBLE|java.lang.B
oolean|true
1087419810.000|ofsSendRecv|1|ClientSeqNo=8|CapTime=1087419810.000|MsgCount=1
1087419810.000|M|S|2|0|30
1087419810.000|P|S|135|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087419810.000|P|S|137|java.awt.Point|0|java.awt.Point[x=706,y=464]
1087419810.000|P|S|139|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087419810.000T|S|1|ServerSeqNo=8|CapTime=1087419810.000|MsgCount=1


1087402349.296|ofsShowWindow|WINDOW_START_APP|11|OFS_ENDMSG|173|PROPERTY_VISIBLE|java.lang.B
oolean|true
1087402349.296|ofsSendRecv|1|ClientSeqNo=8|CapTime=1087402349.296|MsgCount=1
1087402349.296|M|S|2|0|30
1087402349.296|P|S|135|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087402349.296|P|S|137|java.awt.Point|0|java.awt.Point[x=706,y=464]
1087402349.296|P|S|139|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087402349.296|T|S|1|ServerSeqNo=8|CapTime=1087402349.296|MsgCount=1
```

Moving the OFS transaction loop

To enable movement of the QALoad transaction loop in the C++ script, you must first record a full business transaction and a partial business transaction. The business transaction is the activity that you would like to repeat during QALoad playback. Insert QALoad capture comments (using the Insert Command button on the Recording toolbar) at the start and end of a business transaction. These comments will help you find the spots in the script where you would like to reposition the BEGIN_TRANSACTION() and END_TRANSACTION() statements. Then re-start the business transaction.

QALoad's OFS script presents a sequence of Forms GUI objects. The GUI objects contain context dependencies. For example, when a window is opened, the buttons, text fields and edit boxes inside that window are logically dependent on the state of that window. When only one business transaction is captured and the corresponding script's transaction loop is moved, the sequence of the GUI objects is broken during the second iteration of the transaction loop. The broken sequence results in a broken context, which causes the server to respond unpredictably during playback on the second and subsequent iterations of the transaction loop. When the business transaction is restarted during capture, the Forms GUI objects that compose the new transaction are used to anchor into the new transaction loop without breaking the context dependencies of GUI objects.

When modifying the script, use the comment lines as guides in moving the END_TRANSACTION() and BEGIN_TRANSACTION() statements. Ensure that there is a contextual flow from the new position of the END_TRANSACTION() statement to the new position of the BEGIN_TRANSACTION() statement. The set of GUI objects that belong to the ofsSendRecv() statement just before the new END_TRANSACTION() statement must be the same as the set of GUI objects that belong to the ofsSendRecv() statement prior to the new BEGIN_TRANSACTION() statement.

During playback, modify the Conductor setting for Transaction Pacing on the Script Assignment tab to allow the database to process each new business transaction.

The following example shows a modified OFS transaction loop:

New position of the BEGIN_TRANSACTION statement

```
/*
NewSales
*/

DO_SLEEP(13);
ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "B" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 1, 1);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "1" );
ofsIndexKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 175, 97, 0);

DO_SLEEP(6);
ofsSendRecv(1); //ClientSeqNo=31|MsgCount=2|1093981339.921
BEGIN_TRANSACTION();

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "Business World" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 14, 14);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsRemoveFocus( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 174 );
ofsSetSelection( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ADD, 195, 0, 0);
ofsSetCursorPosition( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ENDMSG, 193, "0" );
ofsFocus( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ENDMSG, 174 );

DO_SLEEP(6);
ofsSendRecv(1); //ClientSeqNo=32|MsgCount=4|1093981347.296
```

New position of the END_TRANSACTION statement

```
/*
EndTrans
*/

DO_SLEEP(39);
ofsSendRecv(1); //ClientSeqNo=61|MsgCount=4|1093981458.031

ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsSelectMenuItem( "Sales Orders", 257, OFS_ENDMSG, 477, "MENU_11059" );

DO_SLEEP(26);
ofsSendRecv(1); //ClientSeqNo=62|MsgCount=2|1093981485.265

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "B" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 1, 1);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "1" );
ofsIndexKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 175, 97, 0);

DO_SLEEP(3);
ofsSendRecv(1); //ClientSeqNo=63|MsgCount=2|1093981488.437
END_TRANSACTION();

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "Business World" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 14, 14);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsIndexSKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 176, 10, 0);

DO_SLEEP(13);
ofsSendRecv(1); //ClientSeqNo=64|MsgCount=2|1093981502.640
```

Tips:

During capture, the OFS configuration parameter "record=names" must be enabled to produce control names that may be included in the converted script. Control names persist throughout the Forms session, unlike control IDs, whose values may change at runtime. Add the "record=names" parameter in the `Formsweb.cfg` file or add this parameter to the startup servlet URL. Control IDs can create problems when the transaction loop is moved. Some of the control IDs that have been instantiated by the server prior to the new transaction loop lose context during iterations of the new loop. For example, in a second loop iteration, the server assumes that these client controls are new, generates new control IDs, and eventually cannot find the proper context. Then the server stops responding. If control names are used, Forms objects that have been instantiated before the new transaction loop are maintained through all iterations of the loop because the control name persists throughout the application session.

During playback, ensure that the sleep factor is at 100% and that the transaction pacing is set to a large enough value for the server to process the business transaction that is contained in the new loop. These options can be set on the Script Assignment tab of the Conductor.

OFS and WWW Universal sessions

You can record with a Universal session to capture both the OFS and WWW transactions and merge the two sets of transactions into one script. The captured WWW statements contain non-servlet, non-Forms data such as GIF objects, while the captured OFS statements contain the Forms data.

Universal scripting for OFS-WWW sessions is available in C++ format only. After conversion, the WWW statements do not appear in visual scripts.

Note: The only Universal session combination that is available for Oracle Forms Server is the combination of WWW and Oracle Forms Server.

When an Oracle Applications login is captured, the login can be scripted using the OracleAppsLogin statement or the ofsSetICXTicket statement. Compuware recommends that you use ofsSetICXTicket.

When OracleAppsLogin is used, the login is performed twice: once by the scripted DO_Http statement for the WWW actions and again by OracleAppsLogin. To prevent duplicate logins, you must comment out the DO_Http (WWW middleware) statement.

When ofsSetICXTicket is used, the login is performed just once. This statement allows the WWW login to execute, extracts the ICX ticket from the server reply, and passes the ICX ticket to the Forms session.

To use ofsSetICXTicket, you must modify the script.

To capture an Oracle Applications login with ofsSetICXTicket:

1. Add the following variable declaration statements to the top of the script:
```
char *p;
char ICX_Ticket[100];
char *pTicket;
```
2. In the `*.postcapweb` file, find the HTTP request that returns the ICX ticket. The reply should contain a string that indicates the ICX ticket value, such as `"ICX_TICKET="`. Note the left and right characters that delimit the ICX ticket value. In the example in step 4, the left delimiter is `"icx_ticket='"` and the right delimiter is `"'"`.
3. In the script, find the matching request line for the HTTP request.
4. After the matching HTTP request line, add the DO_GetUniqueString statement using your chosen delimiters. For example:
```
p = DO_GetUniqueString( "icx_ticket='", "'");
```
5. Add script lines that copy the extracted value into your script variables.
```
strcpy(ICX_Ticket, p);
pTicket=ICX_Ticket;
```
6. (optional) Verify the ICX ticket value.
```
RR__printf("ICX_Ticket=\"%s\"\n", ICX_Ticket);
```
7. Add the script line `ofsSetICXTicket(&pTicket);` before the ofsInitSessionCmdLine. This passes the value of the ICX ticket to the ofsInitSessionCmdLine statement.
8. Free the memory allocated by DO_GetUniqueString and ofxSetICXTicket before the end of the transaction:
```
free(p);
p=NULL;
free(pTicket);

pTicket=NULL
```

SAP

Required commands

Certain commands must be present in an SAP script for it to run successfully. These commands are created automatically during the conversion process. Most of the commands exist before the BEGIN_TRANSACTION statement. The required commands include:

```
SET_ABORT_FUNCTION(abort function);

DEFINE_TRANS_TYPE("capture.cpp");

HRESULT hr = CoInitialize(0);

if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info, "ERROR initializing COM");

SAPGuiSetCheckScreenWildcard(‘*’);

SYNCHRONIZE();
```

Required commands for transaction restarting

When transaction restarting is enabled in the Conductor for an SAP script, the following commands, which are automatically added by QALoad during script conversion, must exist for the script to run:

```
SAPGuiApplication(RegisterROT);
SAPGuiApplication(RevokeROT);
SAPGui_error_handler(s_info, buffer);
```

The SAPGuiApplication command properly registers and removes the script's SAP GUI usage on the Runtime Object Table (ROT). If a transaction fails, these actions are taken to start and clean up the SAP environment.

Note: Do not call RR__FailedMsg in an SAP script if the script includes a restart transaction operation. SAPGui_error_handler can be called with the same parameters as RR__FailedMsg to output a fatal error message while still allowing a proper clean up of the current transaction before restarting the transaction.

Error handling and reporting

A try/catch block is automatically generated for the commands between the BEGIN_TRANSACTION and END_TRANSACTION statements. This construct provides error handling and reporting from the script.

```
BEGIN_TRANSACTION();

try{

   SAPGuiConnect( s_info,"qacsapdb2");
   SAPGuiVerCheckStr("6204.119.32");

   //Set SapApplication = CreateObject("Sapgui.ScripingCtrl.1")
   //SapApplication.OpenConnection ("qacsapdb")
   //Set Session = SapApplication.Children(0).Children(0)

   DO_SLEEP(3);

   SAPGuiPropIdStr("wnd[0]");
   SAPGuiCmd3(GuiMainWindow, ResizeWorkingPane, 83, 24, false);

   DO_SLEEP(6);

   SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
   SAPGuiCmd1(GuiTextField,PutText,"qaload1");

   SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
   SAPGuiCmd1Pwd(GuiPasswordField, PutText,"~encr~1211616261");

   SAPGuiCmd0(GuiPasswordField,SetFocus);
   SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

   SAPGuiPropIdStr("wnd[0]");
   SAPGuiCmd1(GuiMainWindow,SendVKey,0);
   SAPGuiCheckScreen("S000","SAPMSYST","SAP");

   ...

   DO_SLEEP(10);

   SAPGuiPropIdStr("wnd[0]/usr/cntlIMAGE_CONTAINER/shellcont/shell/shellcont[0]/shell");
   SAPGuiCmd1(GuiCtrlTree, ExpandNode, "0000000003");
   SAPGuiCmd1(GuiCtrlTree, PutSelectedNode, "0000000004");
   SAPGuiCmd1(GuiCtrlTree, PutTopNode, "Favo");
   SAPGuiCmd1(GuiCtrlTree, DoubleClickNode, "0000000004");
   SAPGuiCheckScreen("SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access");
   SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
   SAPGuiCmd0(GuiButton,Press);
   SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");

} // end try

catch (_com_error e){
   char buffer[1024];
   sprintf (buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
```

```
            (char *)e.Description(), S_task_id);
   RR__FailedMsg(s_info,buffer);

} // end catch

END_TRANSACTION();
```

To include the log on within the transaction loop, move the SAPGuiConnect call inside the try block as shown in the following example:

```
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("capture.cpp");
RESULT hr = CoInitialize(0);

if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info,"ERROR initializing COM");

SAPGuiSetCheckScreenWildcard('*');

SYNCHRONIZE();

BEGIN_TRANSACTION();

try{

   SAPGuiConnect( s_info,"qacsapdb2");
   SAPGuiVerCheckStr("6204.119.32");
   ...
   SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
   SAPGuiCmd0(GuiButton,Press);
   SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");

} // end try

catch (_com_error e){

   char buffer[1024];
   sprintf(buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
           (char *)e.Description(), S_task_id);
   RR__FailedMsg(s_info,buffer);

} // end catch

END_TRANSACTION();
```

To include the log on outside the transaction loop, move the log off section so that it follows the END_TRANSACTION statement. However, ensure that the recording within the transaction loop begins and ends in the same location in the menu system. For example:

```
SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("capture.cpp");

HRESULT hr = CoInitialize(0);

if( hr != ERROR_SUCCESS )
   RR__FailedMsg(s_info,"ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');

SYNCHRONIZE();

SAPGuiConnect( s_info,"qacsapdb2");

SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
SAPGuiCmd1(GuiTextField,PutText,"qaload1");

SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~1211616261");
SAPGuiCmd0(GuiPasswordField,SetFocus);
SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow,SendVKey,0);
SAPGuiCheckScreen("S000","SAPMSYST","SAP");
```

```
BEGIN_TRANSACTION();

try{
   SAPGuiVerCheckStr("6204.119.32");
   ...
} // end try

catch (_com_error e){
   char buffer[1024];
   sprintf(buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
           (char *)e.Description(), S_task_id);
   RR__FailedMsg(s_info,buffer);

} // end catch

END_TRANSACTION();

SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
SAPGuiCmd0(GuiButton,Press);
SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");
```

Handling multiple logons

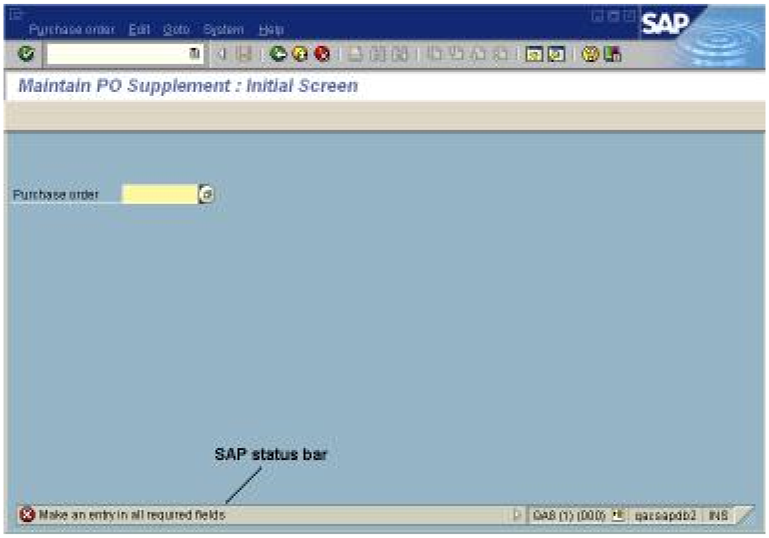
You may need to modify your script to handle multiple logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this issue, you can use the SAPGuiPropIdStrExists and SAPGuiPropIdStrExistsEnd commands to handle either scenario. This technique works by checking for the multiple logon dialog box from SAP and selecting the Continue option.

The following example demonstrates the usage of the SAPGuiPropIdStrExists and SAPGuiPropIdStrExistsEnd commands to handle multiple logons:

```
...

SAPGuiCheckScreen("S000","SAPMSYST","SAP");
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");

   DO_SLEEP(24);

   SAPGuiCmd0(GuiRadioButton,Select);
   SAPGuiCmd0(GuiRadioButton,SetFocus);
   SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");
   SAPGuiCmd0(GuiButton,Press);
   SAPGuiCheckScreen("S000","SAPMSYST","License Information for Multiple Logon");

SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");

...
```

Checking the SAP status bar

The SAP status bar displays error and status messages, as shown in the following figure.



You can use the SAPGuiCheckStatusbar command to test for certain status responses in the SAP environment.

The SAPGuiCheckStatusbar command is used in the following script example:

```
...
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow, SendVKey, 0);
SAPGuiCheckScreen("S000", "SAPMSYST", "SAP");
SAPGuiCmd3(GuiMainWindow, ResizeWorkingPane, 94, 24, false);

//SAPGuiCheckStatusbar returns TRUE if the message is found
//and FALSE if not found

BOOL bRetSts = SAPGuiCheckStatusbar("wnd[0]/sbar", "E: Make an entry in all required
fields");

if (bRetSts)
    RR__printf(" True\n");

else
    RR__printf(" False\n");

...
```

Object life span

Whenever a script is run, all objects on the SAP GUI window are deleted and re-created. These objects, which are created in the SAP environment and can disappear without user interaction, can cause script failure if the script references the objects after they have disappeared.

For more troubleshooting information, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Winsock

Understanding data representation in the script

This section describes how data that is sent and received is displayed in a Winsock script. Use this section as a reference when you examine a script.

During the conversion process, QALoad determines how to represent each character in the script. This conversion process uses the following rules:

1. The character is compared to the “space” character in the ASCII table, which has a decimal value of 32. If the character’s value is less than 32, the following steps are taken:

   b. If the character is “\r”, “\n”, “\t”, or “\f”, it is represented in the script as a normal C escape character.

   c. If the character is either “^\” or “^^”, it is represented in the script as an octal character. For example, the values would be “\034” and “\036”, respectively.

   d. If the character’s value is less than 32 and it does not meet the descriptions in a) and b) above, it is represented in the script as a control character. For example, if the character is a null character, it is represented in the script as “^@”.

2. If the character’s decimal value is between 32 (the “space” character) and 126 (~), it displays in the script as a standard readable ASCII character, with the following exceptions:

   - If the character is “\”, which has a decimal value of 92, it is represented as “\\” in the script.
   - If the character is “““, which has a decimal value of 34, it is represented as “\”” in the script.
   - If the character is “^”, which has a decimal value of 94, it is represented as “^^” in the script.

3. If the character has a decimal value of 127, which corresponds to Delete (DEL), it is represented as “^” in the script.

The following table summarizes the results of rules 1-3.

Code	Octal	Decimal	Char
^@	000	0	NUL
^A	001	1	SOH
^B	002	2	STX
^C	003	3	ETX
^D	004	4	EOT
^E	005	5	ENQ
^F	006	6	ACK
^G	007	7	BEL
^H	010	8	BS
\t	011	9	HT
\n	012	10	LF
^K	013	11	VT
\f	014	12	FF
\r	015	13	CR
^N	016	14	SO
^O	017	15	SI
^P	020	16	SLE

`^Q`	021	17	SC1
`^R`	022	18	DC2
`^S`	023	19	DC3
`^T`	024	20	DC4
`^U`	025	21	NAK
`^V`	026	22	SYN
`^W`	027	23	ETB
`^X`	030	24	CAN
`^Y`	031	25	EM
`^Z`	032	26	SIB
`^[`	033	27	ESC
`\034`	034	28	FS
`^]`	035	29	GS
`^_`	037	31	US
	040	32	SP
`\"`	042	34	"
`\\`	134	92	\
`^^`	136	94	^
`^?`	177	127	DEL

4. If the character is not included in the groups defined in steps 1-3, it is represented as an octal character in the script. These characters are often referred to as high ASCII characters (those with a decimal value greater than 128), and are represented in the script as “\OOO”, where OOO is the octal value for the ASCII character.

Handling Winsock application data flow

Frequently, server programs return unique values (for example, a session ID) that vary with each execution of the script and may be vital to the success of subsequent transactions. To create scripts that include these values, you need to substitute the hard-coded values returned by the server with variables. The following original and modified code examples demonstrate this technique.

Original code

In this script, the server sends a session ID in response to a connection by the client. This session ID is required to successfully complete subsequent transactions.

```
/*
* wsk-AdvancedTechniques_original.c
*
* This script contains support for the following
```

```
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;
{

/* Declare Variables */

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

/////////////////////////////////////////////////////
// The session id returned by the server is
// unique to each connection
/////////////////////////////////////////////////////

/* 21bytes: SessionID=jrt90847\r\n */

DO_WSK_Expect(S1, "\n");

//////////////////////////////////////////////////////
// This unique id is then used for subsequent
// requests
//////////////////////////////////////////////////////

/* 34 bytes */

DO_WSK_Send(S1, "SessionID=jrt90847\r\n:^B^@^@^@^B^@^@^@^A^@^@^@");

/* 15 bytes: ID Accepted#^@\r\n */

DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);

END_TRANSACTION();

REPORT(SUCCESS);
```

```
EXIT();

return(0);
}

void abort_function(PLAYER_INFO *s_info)
{

RR__printf("Virtual User %i:ABORTED.", S_task_id);

EXIT();
}
```

Modified code

If the original script (wsk-AdvancedTechniques_original.c shown above) is replayed, it will fail because the session ID will not be unique; rather, it will be the session ID that is coded in the script. To use the unique session ID received from the server, variable substitution must be used.

```
/*
* wsk-AdvancedTechniques_modified.c
*
* This script contains support for the following
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;

{

/* Declare Variables */

char Buffer[64];
char SendBuffer[64];
int nBytesReceived = 0;

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();
```

```
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

//////////////////////////////////////////////////////
// The reply from the server is read into
// the Buffer variable. We will then have
// the unique Session ID for this connection.
// Also need to null-terminate the buffer
// after receiving.
//////////////////////////////////////////////////////

DO_WSK_Recv(S1, Buffer, 64, 0, &nBytesReceived);
Buffer[nBytesRecieved] = '\0';

/* 21bytes: SessionID=jrt90847\r\n */

//DO_WSK_Expect(S1, "\n");

//////////////////////////////////////////////////////
// Finally, substitute the Session ID received from
// the server with the one coded in the script.
//////////////////////////////////////////////////////

sprintf(SendBuffer, "%s:^B^@^@^@^B^@^@^@^A^@^@^@", Buffer);
DO_WSK_Send(S1, SendBuffer);

/* 34 bytes */

//DO_WSK_Send(S1, "SessionID=jrt90847:^B^@^@^@^B^@^@^@^A^@^@^@");

/* 15 bytes: ID Accepted#^@\r\n */

DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);

}

void abort_function(PLAYER_INFO *s_info)

{
RR__printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
```

Modifying QALoad's functions to incorporate dynamic data

If you need to use dynamic data with your scripts, you can modify some QALoad functions to handle dynamic data. The two scenarios below describe specific situations in which you might need dynamic data, and how to achieve that in the script.

Scenario 1:

One method of accessing dynamic data is by using a datapool file. However, you might need to read in data that is not in the format of an ASCII string, which is required for datapool files.

For example, if the string "\121\101\114\157\141\144" is read in from a datapool file with one of the datapool functions, the output would be "\\121\\101\\114\\157\\141\\144", which is incorrect. To work around this problem, you can use the OctalToChar() command to convert any octal sequences into their binary representation. The following examples illustrates the use of the OctalToChar() command for this purpose:

Example

In this example, the string “\ 121\ 101\ 114\ 157\ 141\ 144” is read in from a central datapool file and converted to its binary representation.

```
/* Declare variables */
char temp[40];

...

BEGIN_TRANSACTION();
GET_DATA();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);

strcpy(temp,VARDATA(1));

OctalToChar(temp); //used to convert octal strings
                   //to their binary format

DO_WSK_Send(S1,temp);
//DO_WSK_Send(S1,"\121\101\122\165\156");
DO_WSK_Closesocket(S1);
```

The DO_WSK_Send() command above sends the string “121\ 101\ 114\ 157\ 141\ 144” to the server. This string is the octal representation of the string “ QALoad ”.

Scenario 2:

You might find that your capture data is not the same data you need for running a test. For example, you might need to change the value of a user ID during replay. One method of changing the value is to change the value through the DO_WSK_Send() command, but that results in the value being static only within the function. To substitute a different value each time, create a dynamic variable, such as a datapool value, to replace the user ID.

Example

In this example, the script includes a DO_WSK_Send() command that sends “name=Jim” to the server as the user ID. Then a variable is used to change the name to “Mark”.

```
/* Declare variables */
char buffer[65];
char sendbuffer[65];

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1,ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//original DO_WSK_Send(S1,"name=Jim");

strcpy( buffer, "Mark");
sprintf( sendbuffer, "name=%s", buffer);
DO_WSK_Send(S1, sendbuffer);

/* 2 bytes: ok */

DO_WSK_Expect(S1,"ok");
DO_WSK_Closesocket(S1);
```

The buffer before the DO_WSK_Send() command is modified and a new buffer is passed as the second parameter of the DO_WSK_Send() command. This effectively sends "name=Mark" to the server instead of "name=Jim".

Saving server replies

There are two methods for saving the entire reply that a server sends back. The following paragraphs describe each method.

Using the Response() and ResponseLength() commands

The Response() command can be called directly after the DO_WSK_Expect() command. It returns a pointer to the data that has been received by DO_WSK_Expect(). To also receive the length of the replay, call the ResponseLength() command, which returns the number of characters that were received. The following example uses the Response() and ResponseLength() commands.

Example

In this example, variables are declared to store the results from the two functions. Both functions are also used to save the buffer that is received within the DO_WSK_Expect() command.

```
/* Declare Variables */
int x = 0;
char *temp;

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 21 bytes: You are now connected */

DO_WSK_Expect(S1, "d");

// Used to store the data that was received by the
// DO_WSK_Expect

temp = Response();

// Used to get the size of the response that was received
// so far

x = ResponseLength();

/* The line below will print the length of the response and the actual response */

RR__printf("length = %d, and response= %s",x, temp);
DO_WSK_Closesocket(S1);
```

The message "length=21 response=You are now connected" displays in the Player buffer window.

Using the DO_WSK_Recv() command

To save a response based on its size instead of a unique character string that is used within the DO_WSK_Expect() command, use the DO_WSK_Recv() command. This command enables you to specify how much data to receive and where to store the data.

You can also use the DO_WSK_Recv() command to store the reply that is returned from the server. This strategy is useful when you need to retrieve the buffer that is returned from the server, even though the returned data is too dynamic and causes the DO_WSK_Expect() command to fail every time.

Example

In this example, the DO_WSK_Recv() command is used to save a server reply based on size. Two variables are declared to store the results from the DO_WSK_Recv() command.

```
/* Declare Variables */
int size = 0;
char temp[45];

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 21 bytes: You are now connected */

memset(temp,'\0',45);
DO_WSK_Recv(S1,temp,45,0,&size);
RR__printf("size=%d string=%s",size,temp);
DO_WSK_Closesocket(S1);
```

The message “size=21 string=You are now connected” displays in the Player buffer window.

Note: If you use this method as a substitute for the DO_WSK_Expect() command, ensure that you receive the correct information prior to calling the next function in the script.

Parsing server replies for values

To parse a buffer for a particular value, you can write a parsing routine that searches the entire buffer for the value. However, you can also use one of QALoad 's Winsock helper commands. The following scenarios describe two situations in which you could use the Winsock commands to solve a parsing problem.

Scenario 1:

To find a string in a server reply, you can use the SkipExpr() and ScanExpr() commands. SkipExpr() searches for the first occurrence of a string in the internal buffer that contains the response that was received within the DO_WSK_Expect() command. Then, use the ScanExpr() command to search for another string. ScanExpr() saves the buffer from the first occurrence of the string that was used with SkipExpr() up to and including the string used within ScanExpr(). The first parameter of ScanExpr() is a UNIX-style regular expression. The following table lists the most common expressions:

Character	Meaning
.	Matches the end of a string.
*	Matches any number of characters.
?	Matches any one character.

Example In this example, the buffer contains “sessionid=1234567890abc”, and the goal is to retrieve everything after the “=”, up to and including “abc”.

```
/* Declare Variables */
char temp[35];
int size = 0;

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 23 bytes: sessionid=1234567890abc */

DO_WSK_Expect(S1, "c");
SkipExpr("sessionid=");
size=ScanExpr(".*abc" , temp);
RR__printf("length = %d string = %s", size, temp);
DO_WSK_Closesocket(S1);
```

The message “length=13 string=1234567890abc” displays in the Player buffer window.

Scenario 2:

You may have data returned from the server that is too dynamic, that is, you are not able to base parsing on actual characters. The solution is to base the parsing on character positions instead.

For example, to save the characters 20 through 25, you could use the ScanSkip() and ScanString() commands. ScanSkip() skips a specified number of characters in the internal buffer that stores the response that was received within the DO_WSK_Expect() command. ScanString() scans a number of characters from the current position within the buffer into a character string.

Example

In this example, a buffer containing “xxx123456789yyy” is returned from the server. The value between “xxx” and “yyy” is returned.

```
/* Declare Variables */

char temp[15];

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 16 bytes: xxx0123456789yyy */

memset(temp,'\0',15);
DO_WSK_Expect(S1, "yyy");
ScanSkip(3);
ScanString(10,temp);
RR__printf("string=%s",temp);
DO_WSK_Closesocket(S1);
```

The message “string=0123456789” displays in the Player buffer window.

WWW

Simulating variable IP addresses

While QALoad can simulate multiple virtual users from a single system, it generally does so using a single source IP address. In most testing situations this isn't a problem, but with a small set of HTTP-based applications, it may not be the best way to simulate real-life activity. For QALoad Player machines with more than one static IP address, QALoad can direct each virtual user to use a different source IP address.

To accomplish this, a local datapool file containing a list of local static IP addresses must be created on each QALoad Player machine. When you enable IP spoofing in the QALoad Conductor, the QALoad Conductor instructs each QALoad Player to create the appropriate datapool file at run time. The QALoad Player uses these addresses for connections to HTTP and SSL servers. Each virtual user receives one address for use with all its connections. If there are more virtual users than addresses, IP addresses are re-used starting from the beginning of the datapool file.

Modifying a Script to Use Variable IP Addresses

QALoad uses the DO_IPSpoofEnable command to insert IP addresses from the datapool into the script. When this command is executed, the script opens the datapool file located on the QALoad Player, reads the first available data record, and stores that record for use on all subsequent DO_Http and DO_Https calls. If there are more virtual users than IP addresses in the datapool file, IP addresses are reused. You can automatically generate the DO_IPSpoofEnable command in your script during conversion by selecting the IP Spoofing option from the QALoad Script Development Workbench's WWW Advanced dialog box. Access this dialog box from the Convert Options wizard's WWW tab by clicking the Advanced button. This option inserts the DO_IPSpoofEnable command directly in the script during conversion, before the first DO_Http or DO_Https command.

Creating a Datapool of IP Addresses

Use the following procedure to create a datapool of valid IP addresses from the QALoad Conductor. This file is automatically created on the QALoad Player workstations (Windows and UNIX) at run time.

To create a datapool of IP addresses:

1. Start *QA*Load Conductor.
2. Click the **Machine Configuration** tab.
3. Double-click the Player machine name in the list. The **Properties** dialog box appears.
4. Select the **Generate IP Spoof Data (machines with multiple IP addresses only)** option.
5. Click **OK**.

At run time, the QALoad Conductor sends a command to each QALoad Player Agent to create the datapool file of IP addresses, and the script is sent to the server using the different IP addresses.

The Generate IP Spoof Data check box is valid only for WWW scripts.

Note: The machine on which the QALoad Conductor resides must have static IP addresses assigned to it. If no static IP addresses are found, the QALoad Conductor displays a warning and the datapool file is not generated. The datapool file is named ipspoof.dat, and is saved in the `\Compuware\QALoad\Datapools` directory.

Handling error messages from the Web server

When a server returns an error message, it returns it in one of two ways. It either returns an error message with a response code (for example, 404 Not Found) or returns an HTML page that contains an error message. The following sections provide examples of code that you can use in your script to handle errors that the Web server returns to the browser.

Handling error messages with response codes

The example below demonstrates how to write code to handle error messages that include response codes that the Web server returns to the browser. The code performs the following actions:

- Checks for an error code using the DO_GetLastHttpError command
- Aborts or continues script execution, based on the WWW_FATAL_ERROR statement

Example

```
int error;
char errorString[30];

DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

if((error = DO_GetLastHttpError()) > 399)

{
sprintf(errorString, "Error in response: %d\n", error);
WWW_FATAL_ERROR("Request-host", errorString);
}
```

Handling error messages returned in an HTML page

The examples below demonstrate how to write code to handle error messages that the Web server returns to the browser in an HTML page.

Using DO_VerifyDocTitle to verify page requests

By inserting the DO_VerifyDocTitle command into your script, you can compare the HTML document titles in your load test script with the document titles you originally captured. The code performs the following actions:

- Calls DO_Http to request an HTML page from the Web server
- Calls DO_VerifyDocTitle with the original HTML document title. If the titles do not match, DO_VerifyDocTitle exits the script

Example

```
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
```

Searching response text for error messages

In some scripts, error messages are displayed as text in an HTML page. The following example demonstrates how to detect these messages in a script. The code performs the following actions:

- Searches for errors returned as HTML from the Web server
- Branches to error handling code

Example

```
int response;
response = DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
if (strstr (response, "200 OK") == NULL)
   WWW_FATAL_ERROR("host", "Response did not have 200 OK");
```

Simulating CGI requests

The following topics describe strategies for simulating CGI requests:

CGI parameter encoding
CGI Get requests

CGI Post requests
CGI forms

Simulating JavaScript

JavaScript is handled by the following process:

1. The browser makes a page request to a server for a page that contains JavaScript.
2. Because JavaScript is simply uncompiled code, the browser downloads and immediately executes this code upon receipt of the page.

Supported objects

QALoad supports the built-in JavaScript objects (global, object, function, array, string, boolean, number, math, date, regexp, and error), document objects, and image objects.

Supported properties

The only document properties that QALoad supports are cookies, title, and the images array. The only image property that QALoad supports is src.

Evaluation errors

If an object, property, or function used within a block of JavaScript code is not defined, it will cause a JavaScript exception. The exception stops evaluation of that block.

Example Web page

The following Web page contains the JavaScript function and an onLoad tag that calls the scrollit function. The onLoad tag tells the browser to execute the JavaScript immediately after loading the page. The scrollit function displays a scrolling banner region on the Web page.

```
<HTML>
<HEAD>
<TITLE>Java Script Example</TITLE></HEAD>

<SCRIPT LANGUAGE="JavaScript" src="js_do_nothing.js">

function scrollit_r2l(seed)
{

var m1 = " Welcome to Compuware's QALoad homepage.";
var m2 = " Glad to see you.";
var m3 = " Thanks for coming. ";
var msg = m1 + m2 + m3;
var out = " ";
var c = 1;

if (seed > 100) {
seed--;
var cmd="scrollit_r2l(" + seed + ")";
timerTwo=window.setTimeout(cmd,100);
}

else if (seed <= 100 && seed > 0) {
for (c=0 ; c < seed ; c++) {
out+=" ";
}
out+=msg;
seed--;
var cmd="scrollit_r2l(" + seed + ")";
window.status=out;
```

```
timerTwo=window.setTimeout(cmd,100);
}
else if (seed <= 0) {
if (-seed < msg.length) {
out+=msg.substring(-seed,msg.length);
seed--;
var cmd="scrollit_r2l(" + seed + ")";
window.status=out;
timerTwo=window.setTimeout(cmd,100);
}
else {
window.status=" ";
timerTwo = window.setTimeout("scrollit_r2l(100)", 75);
}
}
}
</script>
<BODY onLoad="timerONE=window.setTimeout('scrollit_r2l(100)',500);">
<!-- End scrolltext -->

<center><h2>Java Script Example</h2><hr>Check out the browser's scrolling status
 bar.<br><br>
</center>
</BODY></HTML>
```

Example script

The following script features a DO_Http call to retrieve the JavaScript page.

How It Works: QALoad evaluates the JavaScript in the context of script blocks, onLoad tags, and src and then executes them.

```
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
DO_AutomaticSubRequests(TRUE);
...
...
DO_Http("GET http://www.host.com/js.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Java Script Example", TITLE);
...
...
END_TRANSACTION();
```

Simulating cookies

This section describes how QALoad handles cookies. Cookies are handled by the following process:

1. The browser makes a CGI request to a server for a dynamic page.
2. When the server sends the page back to the browser, the page includes a cookie in the header. The browser saves the cookie along with information that ties it to the Web server.
3. On all subsequent requests to that Web server, the browser passes the cookie along with the request.

Example Web page

The following CGI Perl script generates a Set-Cookie header as a part of subsequent HTTP requests.

```
Set-Cookie: SaneID=172.22.24.180-4728804960004
Set-Cookie: SITESERVER=ID=f0544199a6c5970a7d087775f83b23af

<html>

...

The cookies for this site are:<br><br>

<B>SaneID=172.22.24.180-4728804960004; SITESERVER=ID=f0544199a6c5970a7d087775f83b23af
</B><P>

<b>Next cookie for this URL will be : 1</b><br>
<br>RELOAD PAGE TO INCREMENT COUNTER<br><br><A HREF=http://www.host.com/index.htm>Return to
previous homepage.</A>
```

Example script when Dynamic Cookie Handling is turned on

This is the default method by which QALoad handles cookies. The example script features the following elements:

- Two CGI requests that return dynamic pages
- Cookies are handled by the replay engine

```
BEGIN_TRANSACTION();
DO_DynamicCookieHandling(TRUE);

...
...

/* Request: 1 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n\r\n");

/* Request: 2 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n\r\n");

...
...

END_TRANSACTION();
```

Example script when Dynamic Cookie Handling is turned off

The example script features the following elements:

- A CGI request that returns a dynamic page
- Two DO_GetCookieFromReply calls to retrieve the cookie from reply
- Two DO_SetValue calls to set the cookie
- A free cookie

How It Works: For cookies that are set with CGI scripts, the script stores incoming cookies in a variable and passes them back to the Web browser in the reply from the CGI script. The script handles these cookies by executing a DO_GetCookieFromReply command after the CGI request. DO_GetCookieFromReply stores the cookie values in variables, which the script then passes back to subsequent CGI requests using the DO_SetValue command.

```
int i;
char *Cookie[4];

...
...
```

```
for(i=0;i<4;i++)
Cookie[i]=NULL;
DO_InitHttp(s_info);

...
...

BEGIN_TRANSACTION();
DO_DynamicCookieHandling(FALSE);

...
...

/* Request: 1 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl
        "HTTP/1.0\r\n\r\n");

/*Set-Cookie: NUM=1 */
DO_GetCookieFromReplyEx("NUM", &Cookie[0], '*');

/*Set-Cookie: SQUARE=1 */
DO_GetCookieFromReplyEx("SQUARE", &Cookie[1], '*');

/* Request: 2 */
DO_SetValue("cookie000", Cookie[0]); /* NUM=1 */
DO_SetValue("cookie001", Cookie[1]); /* SQUARE=1 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl "
        "HTTP/1.0\r\n"
        "Cookie: {*cookie000}; {*cookie001}\r\n\r\n");

...
...

DO_HttpCleanup();
for(i=0; i<4; i++)
{
free(Cookie[i]);
Cookie[i]=NULL;
}

END_TRANSACTION();
```

Executing a Visual Basic script

QALoad does not evaluate a Visual Basic script. However, any Visual Basic script request that occurs is inserted into the script as a main request.

Executing a Java applet

Java applets are handled by the following process:

1. The browser makes a request to a Web server for an HTML document that contains embedded Java applets.
2. The browser downloads the Java applets, in the order in which they appear on the Web page, and immediately executes them.

Example Web page

The following Web page contains two sections that reference Java applets. Notice the parameters that follow the applet. The browser passes these parameters when invoking an applet.

```
<HTML>
<HEAD>
<TITLE>Java Example</TITLE></HEAD>
<BODY>

<center><h2>Java Applet Example</h2><hr>
```

```
<applet code="LScrollText.class" width="500" height="20" >
<PARAM NAME="MESSAGE" VALUE="Scrolling Text created by Java Applet... >>Click here to
Download<< Use it FREE">
<PARAM NAME="FONTHEIGHT" VALUE="14">
<PARAM NAME="SPEED" VALUE="2">
<PARAM NAME="PIXELS" VALUE="1">
<PARAM NAME="FONTCOLOR" VALUE="0000FF">
<PARAM NAME="BACKCOLOR" VALUE="FFFF00">
<PARAM NAME="TARGET" VALUE="lscrolltext.zip">
</applet>

<br><br><br>

A scrolling message, with custom colors, font size, speed, and target URL.<br>
The source (.ZIP) file can be downloaded by clicking the associated area in text window.

<br><br><br><hr>

<APPLET CODE="imagefader.class" WIDTH=80 HEIGHT=107>
<PARAM name="demicron" value="www.demicron.se">
<PARAM name="reg" value="A00012">
<PARAM name="maxitems" value="3">
<PARAM name="width" value="80">
<PARAM name="height" value="107">
<PARAM name="bitmap0" value="anibal.jpg">
<PARAM name="bitmap1" value="jak.jpg">
<PARAM name="bitmap2" value="jan.jpg">
<PARAM name="url0" value=" ">
<PARAM name="url1" value=" ">
<PARAM name="url2" value=" ">
<PARAM name="step" value="0.05">
<PARAM name="delay" value="20">
<PARAM name="sleeptime" value="2000">

</APPLET>

<br><br><br>

This applet is a very popular image fader that displays a series of images, and allows URLs
to be associated with each image.<br><br><hr>

</center>
</BODY></HTML>
```

Example script

QALoad does not evaluate Java applets. They appear as main requests. The example script features the following elements:

- A DO_Http call to retrieve the main page.
- A DO_Http call to retrieve the scrolling text class.
- A DO_Http call to retrieve the image fader class Java applet.

How It Works: QALoad interacts with the Web server without execution of the Java applet program within the virtual browser. The browser accepts the pages that contain Java applets, but does not execute the applet as part of the load test. The Java applets are not evaluated by QALoad and appear as main requests in the script.

```
DO_InitHttp(s_info);

...
...

BEGIN_TRANSACTION();

...
...
```

```
DO_Http("GET http://www.host.com/java.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Java Example", TITLE);

/* Request: 2 */
DO_Http("GET http://www.host.com/LScrollText.class HTTP/1.0\r\n\r\n");

/* Request: 3 */
DO_Http("GET http://www.host.com/imagefader.class HTTP/1.0\r\n\r\n");
DO_Http("GET http://www.host.com/jak.jpg HTTP/1.0\r\n\r\n");

...
...

END_TRANSACTION();
```

Simulating frames

Frames are handled by the following process:

1. The browser makes a main page request to a Web server for a page that contains frames.
2. The browser parses the frame pages and places them in sub-windows within the browser, each of which displays the frame content.

Example Web page

The following Web page contains four frames.

```
<HTML>
<HEAD>
<TITLE>FRAME Example</TITLE>
</HEAD>

<! -- Here is the FRAME information for browsers with frames -->

<FRAMESET Rows="*,*"><!-- Two rows, each equal height -->
  <FRAMESET Cols="*,*"><!-- Two columns, equal width -->
     <FRAME Src="findex.htm" Name="ul-frame">
     <FRAME Src="findex.htm" Name="ur-frame">
  </FRAMESET>

   <FRAMESET Cols="*,*"><!-- Two columns, equal width -->
     <FRAME Src="findex.htm" Name="ll-frame">
     <FRAME Src="findex.htm" Name="lr-frame">
  </FRAMESET>
</FRAMESET>

</HTML>
```

Example script

QALoad automatically generates all constructs necessary to request frames. The example script features the following element:

- A DO_Http call to retrieve the main page.

How It Works: The frames are treated as sub-requests and are evaluated and requested by QALoad .

```
BEGIN_TRANSACTION();
DO_AutomaticSubRequests(TRUE);

...
...

DO_Http("GET http://www.host.com/frameset.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("FRAME Example", TITLE);

...
...
```

```
END_TRANSACTION();
```

Simulating browser caching

Browser caching is handled by the following process:

1. When the browser makes a request for static HTML pages, it may include an option to retrieve the page only if it is newer than the one held in the browser's cache.
2. If browser caching is enabled, the server returns only newer versions of the page. If browser caching is not enabled, the server always returns the page.

How It Works: The QALoad Script Development Workbench disables browser caching while recording, which means a page is always retrieved.

Requesting password-protected directories

Web developers use password-protected directories to protect access to some pages. When the browser requests a page in a password-protected directory, the server returns a special response that specifies the page is password-protected. When the browser receives this type of reply, it gathers the user ID and password, encrypts them, and passes them back to the server in a subsequent request.

Example script

QALoad automatically generates all the constructs that are necessary to execute a request of a password-protected directory.

The example script features the following elements:

- DO_BasicAuthorization, which takes the user ID and password as parameters
- DO_Http request to the password-protected directory

```
BEGIN_TRANSACTION();
DO_BasicAuthorization("frank", "~encr~557A2549474E57444A");

...
...

DO_Http("GET http://www.host.com/access_controlled/secure.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of a Secured Page", TITLE);

...
...

END_TRANSACTION();
```

Example script

QALoad also handles Windows Domain Authentication (NTLM).

The example script features the following elements:

- A DO_NTLMAuthorization call, which takes the domain, user ID, and password as parameters
- DO_Http request to the NTLM protected directory

```
BEGIN_TRANSACTION();
DO_NTLMAuthorization("dom1\\frank", "~encr~557A2549474E57444A");

...
...

DO_Http("GET http://www.host.com/ntlm_controlled/secure.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of a NTLM Page", TITLE);
```

```
...
...
END_TRANSACTION();
```

Using the WWW Convert Options dialog box

The following topics provide usage tips and resulting script examples for each of the options that are available on the Convert Options dialog box:

Form fields as comments
Anchors as comments
Client image maps as comments
Debug comments
Document title verification
Baud rate
Refresh timeout
Encode CJK characters
Enable Visual Navigator

Advanced options:

Cache
Dynamic redirect handling
Dynamic cookie handling
Automatically process subrequests
Persistent connections during replay
Max concurrent connections
Max connection retries
Server response timeout
HTTP version

Proxy HTTP version
ActiveData
IP spoofing
Streaming media
Hostnames as IP addresses
Strip all cookies from request

Generate XML requests for web services

Traffic filters

Memory Options

Rule Filters

Baud rate

This option is used to simulate slower connections to a Web server, such as 56 Kbps modem or DSL. Specify a baud rate when enabling baud rate emulation in the Convert Options dialog box.

The DO_SetBaudRate command is inserted in the script with the specified baud rate as its only parameter. If baud rate emulation must be asymmetric (upload rate is different than the download rate), use the DO_SetBaudRateEx command. DO_SetBaudRateEx takes two parameters: the upload baud rate and the download baud rate.

Baud Rate (57600) – Yes

The following example has the Baud Rate check box selected and set to 57600.

```
...
...
BEGIN_TRANSACTION();
DO_SetTransactionStart();
DO_SetBaudRate(57600);
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Baud Rate Emulation – No

The following example has the Baud Rate check box cleared.

```
...
...
BEGIN_TRANSACTION();
DO_SetTransactionStart();
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Cache

When this option is enabled, requested images are cached at playback time. The image cache is cleared by the next iteration of the DO_Clear command.

Script example with the Cache option selected

The following example has the Cache option selected.

```
...
...
BEGIN_TRANSACTION();
DO_Cache(TRUE); /* Enable cache */
...
...
END_TRANSACTION();
...
...
```

Script example with the Cache option not selected

The following example has the Cache check box cleared.

```
...
...
BEGIN_TRANSACTION();
DO_Cache(FALSE); /* Disable cache */
...
...
END_TRANSACTION();
...
...
```

CGI forms

CGI forms are handled by the following process.

1. The browser requests a page that contains a CGI form. It displays the page and provides the interaction for input fields that the CGI form specifies.
2. A user enters data into the CGI form and clicks the submit button. This action causes the browser to process the CGI form's action statement.
3. By processing the action statement, the browser gathers all input fields as name value pairs and passes them to a CGI call that the action statement contains.

Example Web page

The following Web page contains a CGI form with:

- An action statement
- Input fields
- Hidden fields

```
<HTML>
<HEAD><TITLE>Forms Example</TITLE>
</HEAD>
<BODY>
<FORM ACTION="http://www.host.com/cgi-bin/perl_9.pl" method=post>
<TABLE>
<TR>
<TD>Name:
<TD><INPUT NAME="name" SIZE="20" MAXLENGTH=20>
<TR>
<TD>Password:
<TD><INPUT TYPE =password NAME="password" SIZE="20" MAXLENGTH=20>
There is a hidden field containing data here: <INPUT TYPE=hidden NAME="hidden" VALUE="This
rocks!">
Here is another hidden field: <INPUT TYPE=hidden NAME="hidden1" VALUE="Web testing is fun">
</FORM>
</BODY>
</HTML>
```

Example script

QALoad automatically generates all the constructs that are necessary to make a CGI form request.

The example includes the following features:

! A DO_Http call to retrieve the forms page.

! Commented description of the input fields on the page.

! GetFormValueByName commands to retrieve the values of the hidden fields from the form.

! DO_SetValue calls to store the field names and their user-entered values.

! A DO_Http call for the CGI get request.

```
char *Field[2];
char *ActionURL[1];

...
...

for(i=0;i<2;i++)
Field[i]=NULL;

for(i=0;i<1;i++)
ActionURL[i]=NULL;

...
...

BEGIN_TRANSACTION();

...
...

/* Request: 1 */

DO_Http("GET http://www.host.com/forms.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Forms Example", TITLE);

/* ActionURL[0]="http://www.host.com/cgi-bin/perl_9.pl" */

DO_GetFormActionStatement(FORM(1), &ActionURL[0]);

/* Form:1 text Name: name, Value: , Desc: */
/* Form:1 text Name: password, Value: , Desc: */
/* Form:1 hidden Name: hidden, Value: This rocks! */

DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1, &Field[0]);

/* Form:1 hidden Name: hidden1, Value: Web testing is fun */

DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1, &Field[1]);

/* Request: 2 From: Forms Example */

DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "form-name");
DO_SetValue("password", "form-password");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name}&{password}");

DO_VerifyDocTitle("Forms Example - Results", TITLE);

...
...

for(i=0; i<2; i++)
{
free(Field[i]);
Field[i]=NULL;
}
```

```
for(i=0; i<1; i++)
{
free(ActionURL[i]);
ActionURL[i]=NULL;
}
END_TRANSACTION();
```

CGI Get requests

Get requests are handled by the following process:

1. The browser makes a request to a server for a URL that contains a call to a CGI program.
2. The server calls the CGI program, which usually returns a Web page. The returned page is referred to as a dynamic page because it is created by the CGI program.
3. The browser accepts the resulting dynamic page and displays it.

Example Web Page

The following Web page contains an anchor (link) that references a CGI program. The reference results in a CGI Get request.

The anchor calls the CGI program named perl_1.pl with some parameters. In perl_1.pl?name=FRANK, the question mark (?) denotes the start of parameters that need to be passed to the program. The name/value pair being passed to the perl_1.pl program is name=FRANK.

When you click the anchor text (dynamic HTML page), the browser makes a CGI Get request. A Get request, when executed by the server, passes parameters in an environment variable to the CGI program. This type of parameter handling is limited to 255 characters.

```
<HTML>
<HEAD>
<TITLE> QALoad WWW Capture Examples</TITLE>
</HEAD>
<BODY>
<A HREF="/cgi-bin/perl_1.pl?name=FRANK">Dynamic HTML Page</A>
</BODY>
</HTML>
```

Example Script

QALoad automatically generates all constructs that are necessary for a CGI Get request. The following script uses a DO_Http call for the CGI Get request.

How It Works: The script processes a CGI Get request in the same way as it processes URL links to a page. In the example script below, note that the parameters passed to the Web server on the CGI call are recorded unchanged. The parameters do not change unless the page is dynamically generated.

```
char *Anchor[1];
for(i=0;i<1;i++)
Anchor[i]=NULL;

DO_InitHttp(s_info);

SYNCHRONIZE();
BEGIN_TRANSACTION();

...

...

DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

/*
* Anchor 'http://www.host.com/cgi-bin/perl_1.pl?name=FRANK'
```

```
* 'Dynamic HTML Page'
*/

DO_GetAnchorHREF("Dynamic HTML Page", &Anchor[0]);
DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Perl Example Page", TITLE);

...

...

for(i=0; i<1; i++)

{
free(Anchor[i]);
Anchor[i]=NULL;
}

END_TRANSACTION();
```

CGI parameter encoding

CGI (Common Gateway Interface) is widely used on World Wide Web sites to provide the ability to run server-side scripts that can take variable input from a Web browser. QALoad recognizes when the browser has communicated to a CGI site and automatically creates variables for parameters whenever necessary. For example, many CGI submission forms contain hidden parameters that the user cannot modify, but are always sent in the WWW request. Because these values can contain variable data, QALoad inserts statements into the script to store these hidden parameters in variables and append them automatically to CGI requests.

CGI requests also include parameters that the browser has allowed the user to modify. For example, a CGI form might require a user to enter a name and address and click a submit button to continue. QALoad does not automatically store these types of parameters in variables, but instead provides an easy way to modify the content of the parameters that are being sent in the CGI request via the DO_SetValue command.

When you modify parameters that are passed into a QALoad CGI request, ensure that all CGI parameters that contain characters that are not alphanumeric (a-Z, 0-9) are encoded prior to being sent to the server. CGI encoding entails inserting the ASCII value of a character, prefixed with the "%" character, into the parameter. QALoad automatically CGI-encodes any values that it detects during the recording and conversion process; however, to manually add or modify any CGI parameter strings after your script is created, you must manually encode special characters to ensure that the CGI parameter data is sent to the Web server properly. For example, to insert the "=" character into a CGI parameter, first determine its ASCII hexadecimal value (3D), and insert that value into the CGI parameter prefixed with "%". In the CGI parameter string, "%3D" would replace "=". All CGI parameter encoding is handled by this method, except for spaces. Blank spaces must be specified in the encoded CGI string by the character "+", rather than the ASCII value.

QALoad provides an automatic way of performing this encoding via the DO_EncodeString command.

CGI Post requests

Post requests are handled by the following process:

1. The browser makes a request to a server for an HTML page that contains a form that uses an action statement with a Post call to a CGI program.
2. When you click the Submit button on a CGI form, the browser makes a Post request and the server returns a Web page.
3. The browser accepts the dynamic page and displays it. Because it is a CGI Post request, the browser passes the parameters of the program to the CGI script as command line options.

Example Web Page

The following Web page contains a form that calls a CGI script with a Post Request.

```
<html>
<head><title> QALoad's Perl Example Page</title>
</head><body><center> QALoad's Perl Example Page</center>
<form name = myform method = POST action = perl_1.pl>
<input type = text name = yourname size = 50><br>
<input type = submit value = "Submit Request">
<input type = reset>
</form>
</body></html>
```

Example Script

QALoad automatically generates all the constructs that are necessary for a CGI Post request. The following script features a DO_HTTP request that executes a CGI Post request :

```
char *ActionURL[1];
...
...
for(i=0;i<1;i++)
ActionURL[i]=NULL;
...
...
BEGIN_TRANSACTION();
/* Request: 1 */
DO_SetValue("name", "FRANK");
DO_Http("GET http://www.host.com/cgi-bin/perl_1.pl?{name} "
        "HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle(" QALoad's Perl Example Page", TITLE);
/* ActionURL[0]="http://www.host.com/cgi-bin/perl_1.pl" */
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);

...

/* Request: 2 From: QALoad's Perl Example Page */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("yourname", "PostFrank");
DO_SetValue("function", "View the log of previous visitors.");
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{yourname}&{function}");

DO_VerifyDocTitle(" QALoad's Perl Example Page", TITLE);

...
...

for(i=0; i<1; i++)
{
free(ActionURL[i]);
ActionURL[i]=NULL;
}

END_TRANSACTION();
```

Debug comments

When this option is selected, some items, such as received replies are placed in comment blocks in the script.

Script example with the Debug comments option selected

The following example has the Debug Comments check box selected.

```
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
/* Received reply: < QALoad WWW Capture Examples> */
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
...
...
```

Script example with the Debug Comments option not selected

The following example has the Debug Comments check box cleared.

```
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
...
...
```

Enable Visual Navigator

The Enable Visual Navigator option enables the Visual Navigator. The Visual Navigator renders your recorded C-based transaction in a tri-paned, browser-like environment similar to popular visually-oriented development tools, with icons that represent all the elements of your script.

Script example with the Enable Visual Navigator option selected

When the Enable Visual Navigator check box is selected, the following conversion options are not available because they do not apply to Visual Navigator:

- Comment Options
- Form field as comments
- Anchors as comments
- Client Image Maps as comments
- Debug comments
- Dynamic Redirect Handling
- Dynamic Cookie Handling
- Automatically Process Sub-Requests
- ActiveData
- IP Spoofing
- Hostnames as IP Addresses

Script example with the Enable Visual Navigator option not selected

If you do not select the Enable Visual Navigator check box, QALoad generates a standard C script. All normal and advanced conversion options apply to the script.

Hostnames as IP addresses

When this option is selected, DO_Http and DO_Https requests include IP addresses for the requests instead of hostnames. This option only works if the hosts can be reached directly. If a host must be reached through a proxy, the IP address cannot be determined.

Script example with the Hostnames as IP Addresses options selected

The following example has the Hostnames as IP Addresses check box selected.

```
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://172.22.24.39/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
...
...
END_TRANSACTION();
...
...
```

Script example with the Hostnames as IP Addresses option not selected

The following example has the Hostnames as IP Addresses check box cleared.

```
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
...
...
END_TRANSACTION();
...
...
```

HTTP version

A WWW script can be set to 1.1 or 1.0. When set to 1.1, all requests and subrequests are sent as HTTP/1.1. When set to 1.0, all HTTP requests and subrequests are sent as HTTP/1.0.

The following example has the HTTP Version field set to 1.0.

```
...
...
BEGIN_TRANSACTION();
DO_HTTPVersion("1.0");
...
...
END_TRANSACTION();
...
...
```

IP spoofing

By default, QALoad searches for the file ipspoof.dat in the Datapools directory. You can override this behavior by providing a different file name as the parameter to the DO_IPSpoofEnable command in the converted script. For example:

```
DO_IPSpoofEnable("myaddresses.dat");
```

Script example with the IP Spoofing option selected

The following example has the IP Spoofing check box selected.

```
...
...
DO_InitHttp(s_info);
DO_IPSpoofEnable("");
...
...
BEGIN_TRANSACTION();
...
...
END_TRANSACTION();
...
...
```

Script example with the IP Spoofing option not selected

The following example has the IP Spoofing check box cleared.

```
...
...
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
...
...
END_TRANSACTION();
...
...
```

Max concurrent connections

This field indicates the maximum number of connections that a DO_Http or DO_Https command will open to the server at any time. These simultaneous connections are only used if sub-requesting is enabled.

The following example has the Max Concurrent Connections field set at 4.

```
...
...
BEGIN_TRANSACTION();
DO_SetTransactionStart();
DO_SetMaxBrowserThreads(4);
...
...
```

```
/* Request: 1 */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
END_TRANSACTION();
...
...
```

Max connection retries

This field specifies the number of times during replay that QALoad will attempt to connect to the server after timing out. The following example has the Max Connection Retries field set at 4.

```
...
...
BEGIN_TRANSACTION();
...
...
DO_SetMaximumRetries(4);
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
```

Persistent connections during replay

This is an option placed in the script to be used at replay time. During replay, QALoad attempts to keep the connection to the Web server open for each DO_Http request that is sent to the server.

Script example with the Persistent Connections During Replay option selected

The following example has the Persistent connections during replay check box selected.

```
...
...
BEGIN_TRANSACTION();
DO_UsePersistentConnections(TRUE);
...
...
END_TRANSACTION();
...
...
```

Script example with the Persistent Connections During Replay option not selected

The following example has the Persistent connections during replay check box cleared.

```
...
...
```

```
BEGIN_TRANSACTION();
DO_UsePersistentConnections(FALSE);

...
...

END_TRANSACTION();

...
...
```

Reuse SSL session ID

This option is available only on an SSL installation of QALoad . By default, this option is not selected and SSL session IDs are not re-used, which reflects standard browser behavior. If your application re-uses SSL session IDs, consider selecting this option.

The Reuse SSL session ID option is used by the replay engine at replay time and the current session's ID is re-used for all the requests within the transaction.

Script example with the Reuse SSL Session ID option selected

The following example has the Reuse SSL session ID check box selected.

```
...
...

SYNCHRONIZE();

/* Select following statement for reuse of Session ID with */
/* SSL. If session ID needs only to be reused within */
/* a transaction insert after the BEGIN_TRANSACTION */
/* statement */

/* DO_SSLReuseSession(TRUE); */

BEGIN_TRANSACTION();

...
...

/* Request: 1 */
DO_Http("GET http://www.host.com/subs.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Page Of Subs", TITLE);

...
...

END_TRANSACTION();

...
...
```

Script example with the Reuse SSL Session ID not selected

The following example has the Reuse SSL session ID check box cleared.

```
...
...

SYNCHRONIZE();

/* Select following statement for reuse of Session ID */
/* with SSL. If session ID needs only to be reused within */
/* a transaction, insert after the BEGIN_TRANSACTION */
/* statement */
```

```
/* DO_SSLReuseSession(FALSE); */
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/subs.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Page Of Subs", TITLE);
...
...
END_TRANSACTION();
...
...
```

Server response timeout

This field specifies, in seconds, the length of time during replay that QALoad will wait for data from the server before timing out. The following example has the Server Response Timeout field set at 120.

```
...
...
DO_InitHttp(s_info);
DO_SetTimeout(120); /* Maximum time to wait for HTTP Reply */
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
```

Strip all cookies from request

When this option is selected, no cookies will be sent in the DO_Http requests.

Script example with the Strip All Cookies From Request option selected

The following example has the Strip All Cookies From Request check box selected.

```
...
...
DO_Http("GET http://qawebserv.compuware.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
...
...
```

Script example with the Strip All Cookies From Request option not selected

The following example has the Strip All Cookies From Request check box cleared.

```
...
...
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n Cookie:
username=username\r\n\r\n");
...
...
```

Traffic filters

The Traffic Filters dialog box enables you to determine which traffic should be included or blocked from your script. This dialog box is accessed by the Traffic Filters button on the WWW Advanced dialog box.

Script example with a traffic filter set

In the following example, a traffic filter has been set to exclude requests with URLs that contain the string "www.host.com".

```
...
...
DO_InitHttp(s_info);
...
...
/* Exclude requests with URLs containing */
DO_BlockRequestsFrom("www.host.com;");
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://rm.host.com:8099/ramgen/realmp3.mp3 "
        "HTTP/1.0\r\n"
        "Referer: http://www.host.com/index.htm\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Script example without a traffic filter set

The following example does not use traffic filters.

```
...
...
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
DO_Http("GET http://www.host.com/index.htm HTTP/ 1.0\r\n\r\n");
```

```
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_Http("GET http://rm.host.com:8099/ramgen/realmp3.mp3 HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Anchors as comments

Anchors as comments

When this option is selected, all anchors are placed in comment blocks in the script.

Example Web page
Script example with option selected
Script example with option not selected

Example Web Page

```
<HTML>
<HEAD>
<TITLE> QALoad WWW Capture Examples</TITLE>
</HEAD>
<BODY>
<HR>
<IMG SRC="../logo.gif" ALIGN=LEFT width="127" height="129">
<IMG SRC="../logo.gif" ALIGN=RIGHT width="127" height="129">
<BR><BR>
<CENTER><H2><EM> QALoad WWW Capture Examples</EM></H2>
<A HREF=../default.htm><h3>Return to welcome homepage.</h3></A>
<BR></CENTER><HR>
<CENTER>
<TABLE CELLSPACING="10">
<TR>
<TH ALIGN=left>LINK:
<TH ALIGN=left>DESCRIPTION:
<TR>
<TD><A HREF="../standard.htm">Standard HTML Homepage</A></TD>
<TD>Static page w/ images (.GIF 87a, 89a), sound files (.WAV), and assorted links</TD>
<TR>
<TD><a href="../subs.htm">Multiple Inline Images Page</a></TD>
<TD>Static page with 16 inline images</TD>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Script example with the Anchors as comments option selected

The following example has the Anchors as comments check box selected.

```
...
...
```

```
/* Declare Variables */

int i;

char *Anchor[1];

...
...

for(i=0;i<1;i++)

Anchor[i]=NULL;

...
...

BEGIN_TRANSACTION();

...
...

/* Request: 1 */

DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);

/* Anchors 'http://www.host.com/default.htm' 'Return to welcome homepage.' */
/* Anchors 'http://www.host.com/standard.htm' 'Standard HTML Homepage' */

DO_GetAnchorHREF( "Standard HTML Homepage", &Anchor[0]);

/* Anchors 'http://www.host.com/subs.htm' 'Multiple Inline Images Page' */

/* Request: 2 To: Standard HTML Homepage From: QALoad WWW Capture Examples */

/* Variable: Anchor000 links to: Standard HTML Homepage on page: QALoad WWW Capture Examples
*/

DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("\"Standard HTML Example\"", TITLE);

...
...

for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
```

Script example with the Anchors as comments option not selected

The following example has the Anchors as comments check box cleared.

```
...
...

/* Declare Variables */
int i;
char *Anchor[1];

...
...

for(i=0; i<1; i++)
Anchor[i]= NULL;

...
...

BEGIN_TRANSACTION();
```

```
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_GetAnchorHREF( "Standard HTML Homepage", &Anchor[0]);

/* Request: 2 To: Standard HTML Homepage From: QALoad WWW Capture Examples */
/* Variable: Anchor000 links to: Standard HTML Homepage on page: QALoad WWW Capture Examples
*/

DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");

...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
```

Automatically process subrequests

Automatically process subrequests

This is a playback option. When this option is selected, subrequests (such as .jpg, .gif, .css, and .js) are not included in the script during conversion, and playback makes the requests at run time.

Example Web page
Script example with option selected
Script example with option not selected

Example Web page

```
<html>
<head>
<title>Page Of Subs</title>
</head>
<body>
<p>The page of subrequests</p>
<p>.<img border="0" src="win2000.gif" width="456" height="124"><img border="0"
src="APACHE.GIF" width="259" height="32"><img border="0" src="banner.gif" width="470"
height="64"><img border="0" src="COLORS.GIF" width="200" height="100"></p>
</body>
</html>
```

Script example with the Automatically Process Subrequests option selected

The following example has the Automatically Process Subrequests check box selected.

```
...
...
BEGIN_TRANSACTION();
DO_AutomaticSubRequests(TRUE);

...
...
DO_Http("GET http://www.host.com/subs.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Page Of Subs", TITLE);

...
...
```

```
END_TRANSACTION();

...
...
```

Script example with the Automatically Process Subrequests option not selected

The following example has the Automatically Process SubRequests check box cleared.

```
...
...

BEGIN_TRANSACTION();

...
...

DO_AutomaticSubRequests(FALSE);

...
...

DO_Http("GET http://www.host.com/subs.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Page Of Subs", TITLE);

/* Request: 2 From: Page Of Subs */
DO_Http("GET http://www.host.com/win2000.gif HTTP/1.0\r\n\r\n");

/* Request: 3 From: Page Of Subs */
DO_Http("GET http://www.host.com/web.gif HTTP/1.0\r\n\r\n");

/* Request: 4 From: Page Of Subs */
DO_Http("GET http://www.host.com/APACHE.GIF HTTP/1.0\r\n\r\n");

/* Request: 5 From: Page Of Subs */
DO_Http("GET http://www.host.com/COLORS.GIF HTTP/1.0\r\n\r\n");

...
...

END_TRANSACTION();

...
...
```

Client image maps as comments

Client image maps as comments

When this option is selected, all client image maps are placed within comment blocks in the script.

Example Web page
Script example with option selected
Script example with option not selected

Example Web Page

```
<html><head></head><body>

<center><h2>Client-side version of clickable imagemap</h2>

Click on one of the fields contained in the image to access the associated link.

<MAP NAME="title">
<AREA SHAPE="rect" COORDS="1,108,115,124" HREF="sup.htm"></AREA>
<AREA SHAPE="rect" COORDS="119,107,234,124" HREF="reg.htm"></AREA>
<AREA SHAPE="rect" COORDS="235,107,352,124" HREF="fea.htm"></AREA>
<AREA SHAPE="rect" COORDS="353,107,466,124" HREF="tech.htm"></AREA>
<AREA SHAPE="rect" COORDS="0,127,156,144" HREF="htmlsam.htm"></AREA>
<AREA SHAPE="rect" COORDS="157,127,312,144" HREF="exampro.htm"></AREA>
```

```
<AREA SHAPE="rect" COORDS="313,127,466,144" HREF="feedback.htm"></AREA>
<AREA SHAPE="rect" COORDS="0,0,466,143" HREF="invalid.htm"></AREA>
</MAP>

<P>

<IMG SRC="title.gif" BORDER="0" ALT="[Netscape FastTrack Server 2.0]" USEMAP="#title"
width="468" height="145">

</center>
</body></html>
```

Script example with the Client Image Maps as Comments option selected

The following example has the Client Image Maps as Comments check box selected.

```
...
...
for(i=0;i<1;i++)
ClientMapURL[i]= NULL;
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/ismap.htm HTTP/1.0\r\n\r\n");
/* Client Map:1 Region:1 HREF: http://www.host.com/sup.htm */
/* Client Map:1 Region:2 HREF: http://www.host.com/reg.htm */
DO_GetClientMapHREF(MAP(1), REGION(2), &ClientMapURL[0]);
/* Client Map:1 Region:3 HREF: http://www.host.com/fea.htm */
/* Client Map:1 Region:4 HREF: http://www.host.com/tech.htm */
/* Client Map:1 Region:5 HREF: http://www.host.com/htmlsam.htm */
/* Client Map:1 Region:6 HREF: http://www.host.com/exampro.htm */
/* Client Map:1 Region:7 HREF: http://www.host.com/feedback.htm*/
/* Client Map:1 Region:8 HREF: http://www.host.com/invalid.htm */
/* Request: 2 */
DO_SetValue("ClientMap000", ClientMapURL[0]);
DO_Http("GET {*ClientMap000} HTTP/1.0\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(ClientMapURL[i]);
ClientMapURL[i]=NULL;
}
END_TRANSACTION();
...
```

Script example with the Client Image Maps as Comments options not selected

The following example has the Client Image Maps as Comments check box cleared.

```
...
...
for(i=0; i<1; i++)
ClientMapURL[i]= NULL;
...
...
/* Request: 1 */
```

```
DO_Http("GET http://www.host.com/ismap.htm HTTP/1.0\r\n\r\n");
DO_GetClientMapHREF(MAP(1), REGION(2), &ClientMapURL[0]);

/* Request: 2 */

DO_SetValue("ClientMap000", ClientMapURL[0]);
DO_Http("GET {*ClientMap000} HTTP/1.0\r\n\r\n");

...
...

for(i=0; i<1; i++)
{
free(ClientMapURL[i]);
ClientMapURL[i]= NULL;
}

END_TRANSACTION();

...
...
```

Document title verification

Document title verification

There are three supported methods for verifying a document title, which are shown in the following sections. Document title verification can be a good tool for detecting and handling error messages that are returned in an HTML page. You can verify a document title by searching for the Entire Document Title, the Prefix of the document title, or the Suffix of the document title. The following is an example of a search for the Entire document title.

Example Web page
Script examples with option selected
Script examples with option not selected

Example Web Page

```
<HTML>
<HEAD>
<title>Welcome to The Main Page</title>
</head>
<body>
<p>

<A href="index.htm">WWW Capture Examples</A> (relative link)

</p>
</body>
</HTML>
```

Script example with the Document Title Verification option selected

The following example has the Document Title Verification check box selected and compares by the entire document title.

```
...
...

DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);

...
...
```

Script example that matches the first 5 characters

The following example compares by prefix.

```
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Welco", PREFIX);
DO_SetTransactionCleanup();
...
...
```

Script example that matches the last 4 characters

The following example compares by suffix.

```
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Page", SUFFIX);
DO_SetTransactionCleanup();
...
...
```

Document Title Verification – No

The following example has the Document Title Verification check box cleared.

```
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
DO_SetTransactionCleanup();
...
...
```

Dynamic cookie handling

Dynamic cookie handling

This is a playback option. When this option is selected, the script does not have any cookie-specific information and playback deals with dynamic cookies at run time.

Example Web page
Script example with option selected
Script example with option not selected

Example Web page

The cookies for this site are:

```
Set-Cookie: SaneID=172.22.24.180-4728804960004
Set-Cookie: SITESERVER=ID=f0544199a6c5970a7d087775f83b23af
```

```
<html>
<head></head>
<body>
<br>RELOAD PAGE TO INCREMENT COUNTER<br><br>
</body>
</html>
```

Script example with the Dynamic Cookie Handling option selected

The following example has the Dynamic Cookie Handling check box cleared.

```
...
...
BEGIN_TRANSACTION();
DO_DynamicCookieHandling(TRUE);
...
...
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Script example with the Dynamic Cookie Handling option not selected

The following example has the Dynamic Cookie Handling check box cleared.

```
...
...
char *Cookie[4];
...
...
for(i=0;i<2;i++)
Cookie[i]=NULL;
...
...
BEGIN_TRANSACTION();
DO_DynamicCookieHandling(FALSE);
...
...
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n\r\n");
/*Set-Cookie: NUM=1 */
DO_GetCookieFromReplyEx("NUM", &Cookie[0], '*');
/*Set-Cookie: SQUARE=1 */
DO_GetCookieFromReplyEx("SQUARE", &Cookie[1], '*');
/* Request: 2 */
DO_SetValue("cookie000", Cookie[0]); /* NUM=1 */
DO_SetValue("cookie001", Cookie[1]); /* SQUARE=1 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl HTTP/1.0\r\n Cookie: {*cookie000};
{*cookie001}\r\n\r\n");
...
...
DO_HttpCleanup();
for(i=0; i<2; i++)
{
free(Cookie[i]);
Cookie[i]=NULL;
}
END_TRANSACTION();
...
...
```

Dynamic redirect handling

Dynamic redirect handling

This is a playback option. When Dynamic redirect handling is enabled, playback automatically handles the redirection.

Consider a Web page with a link to 'Redirected Webpage' (http://www.host.com/cgi-bin/dynredir.exe). When this link is clicked, the server generates a 302 return value with a new redirected location of http://172.22.24.39/cgi-bin/aperl_8.pl.

Script example with option selected
Script example with option not selected

Script example with the Dynamic Redirect Handling option selected

When this option is selected, the script only contains the request for http://www.host.com/cgi-bin/dynredir.exe and replay handles the 302 return value and calls http://172.22.24.39/cgi-bin/aperl_8.pl automatically.

The following example has the Dynamic Redirect Handling check box selected.

```
...
...
/* Declare Variables */
int i;
char *Anchor[1];
...
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_DynamicRedirectHandling(TRUE);
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/""1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
...
...
DO_GetAnchorHREF( "Redirected Webpage", &Anchor[0]);
...
...
DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of Dynamic Redirect Sample", TITLE);
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
```

```
END_TRANSACTION();

...
...
```

Script example with the Dynamic Redirect Handling option not selected

In this example, the script contains the request for http://www.host.com/cgi-bin/dynredir.exe as well as http://172.22.24.39/cgi-bin/aperl_8.pl. The following example has the Dynamic Redirect Handling check box cleared.

```
...
...

/* Declare Variables */
int i;
char *Anchor[1];

...
...

for(i=0;i<1;i++)
Anchor[i]=NULL;

...
...

BEGIN_TRANSACTION();
DO_DynamicRedirectHandling(FALSE);

...
...

DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_GetAnchorHREF( "Redirected Webpage", &Anchor[0]);
DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Document Moved", TITLE);
DO_Http("GET http://www.host.com/redir/frm.pl HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of Dynamic Redirect Sample", TITLE);
DO_SetTransactionCleanup();

/* Clear up some internal storage used for DO_SetValue() */
DO_HttpCleanup();
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}

END_TRANSACTION();

...
...
```

Encode DBCS characters

Encode CJK characters

When this option is selected, the double-byte characters used for Chinese, Japanese, and Korean (CJK) scripts are converted into octal format. Since CJK characters use Double Byte Character Sets (DBCS), encoding must be enabled for a capture with CJK characters, so that the double-byte characters can be viewed in a legible format.

Example Web page
Example script with option selected
Example script with option not selected

Example Web page

```
<html>
<head>
<title>야후! 코리아 </title>
<meta http-equiv="Content-type" content="text/html; charset=euc-kr">
<meta http-equiv="Cache-Control" content="no-cache">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Expires" content="Wed, 04 Jul 1973 16:00:00 GMT">

<!--CSS-->
<style type='text/css'>
</style>

<!--/CSS-->
</head>
<body onload="document.search.p.focus();" topmargin=8>

...
...

</body>
</html>
```

Script example with the Represent CJK as Octal Characters option selected

The following example has the Represent CJK as Octal Characters check box selected.

```
...
...

/* Request: 1 */
DO_Http("GET http://kr.yahoo.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("\276\337\310\304!\304\332\270\256\276\306", TITLE);

...
...
```

Script example with the Represent CJK as Octal Characters option not selected

The following example has the Represent CJK as Octal Characters check box cleared.

```
...
...

/* Request: 1 */
DO_Http("GET http://kr.yahoo.com/ HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle( "야후! 코리아", TITLE);

...
...
```

Form field as comments

Form field as comments

When this option is selected, all forms and their fields are placed in comment blocks in the script.

Example Web page
Script example with option selected
Script example with option not selected

Example Web page

```
<!DOCTYPE HTML PUBLIC "-//AdvaSoft//DTD HTML 3.2 extended 961018//EN">
<HTML>
<HEAD>
  <TITLE>Forms Example</TITLE>
</HEAD>

<BODY BGCOLOR="#7093DB">
<H2 ALIGN=center>Example of HTTP Forms</H2>
<BR>
<FORM ACTION="/cgi-bin/perl_9.pl" method=post>
<TABLE>

<TR>
<TD>Name:
<TD><INPUT NAME="name" SIZE="20" MAXLENGTH=20>

<TR>
<TD>Password:
<TD><INPUT TYPE =password NAME="password" SIZE="20" MAXLENGTH=20>

<TR>
<TD>E-Mail Address:
<TD><INPUT NAME= "e-mail" SIZE = "40" MAXLENGTH=80>
<TR>
<TD>Address:
<TD><INPUT TYPE = text NAME = "Address" SIZE = "40" MAXLENGTH=40>

<TR>
<TD>City:
<TD><INPUT NAME="city" SIZE="40" MAXLENGTH=40>
<TD ALIGN=left>State:
<TD ALIGN=left><INPUT NAME="state" SIZE="2" MAXLENGTH=2 ALIGN=left>
<TD ALIGN=left>Zip:
<TD ALIGN=left><INPUT NAME="zip" SIZE="5" MAXLENGTH=5>

<TR>
<TD VALIGN=top>Favorite Color:
<TD><SELECT NAME="options">
<OPTION>Red
<OPTION>Orange
<OPTION>Yellow
<OPTION>Green
<OPTION selected=on>Blue
<OPTION>Indigo</OPTION>
<OPTION>Violet</OPTION>
</SELECT>

<TR>

<TR>
<TD VALIGN=top>Color of your money:
<TD><SELECT NAME="dates" multiple="multiple">
<OPTION selected=on>Red
<OPTION>Blue
<OPTION>Green</OPTION>
<OPTION>Beige</OPTION>
</SELECT>

<TR>
<TD VALIGN=top>Comments:
<TD><TEXTAREA NAME="comments" COLS=40 ROWS=5></TEXTAREA>
</TABLE>

<BR><INPUT TYPE=checkbox CHECKED NAME="echo">Echo a copy of the result HTML Page to E-
mail<BR><BR>
```

```
<P>

<TABLE>

<TR>
<TD VALIGN=top>Testing:
<TD><INPUT TYPE=radio CHECKED NAME="test" VALUE="capture">Capture<BR>
<INPUT TYPE=radio NAME="test" VALUE="replay">Replay<BR>
<INPUT TYPE=radio NAME="test" VALUE="loadtest">Loadtest<BR>

<TR>

<TR>
<TD>Web page to append to reply:
<TD><INPUT TYPE=file NAME="web page">
</TABLE>

<BR>

There is a hidden field containing data here: <INPUT TYPE=hidden NAME="hidden" VALUE="This
rocks!">
Here is another hidden field: <INPUT TYPE=hidden NAME="hidden1" VALUE="Web testing is fun">

<BR><BR>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="colors.gif" width="200" height="100">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="eye.gif" width="80" height="60">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="devplatform.gif" width="48" height="43">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="enterprise_sm.gif" width="42" height="41">
</TABLE>

<BR>

<TABLE ALIGN=center>
<TR>
<TD><INPUT TYPE=submit NAME="submit">
<TD><INPUT TYPE=reset>
</TABLE>

</FORM>

</BODY>
</HTML>
```

Script example with Form field as comments option selected

The following example has the Form field as comments check box selected.

```
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
...
...
for(i=0; i<2; i++)
Field[i]= NULL;
for(i=0; i<1; i++)
ActionURL[i]= NULL;
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/forms.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Forms Example", TITLE);
/* ActionURL[0]="http://www.host.com/cgi-bin/perl_9.pl" */
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
/* Form:1 text Name: name, Value: */
/* Form:1 text Name: password, Value: */
/* Form:1 text Name: e-mail, Value: */
/* Form:1 text Name: Address, Value: */
/* Form:1 text Name: city, Value: */
/* Form:1 text Name: state, Value: */
/* Form:1 text Name: zip, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 text Name: comments, Value: */
/* Form:1 checkbox Name: echo, Value: */
/* Form:1 radio Name: test, Value: capture, */
/* Form:1 radio Name: test, Value: replay, */
/* Form:1 radio Name: test, Value: loadtest, */
/* Form:1 *unknown* Name: web page, Value: , */
/* Form:1 hidden Name: hidden, Value: This rocks!, */
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1, &Field[0]);
/* Form:1 hidden Name: hidden1, Value: Web testing is fun */
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1, &Field[1]);
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 submit Name: submit, Value: */
/* Form:1 *unknown* Name: , Value: */
/* Request: 2 From: Forms Example */
```

```
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name}&{password}&{e-mail}&{Address}&{city}&{state}&"
        "{zip}&{options}&{dates}&{comments}&{echo}&{test}&"
        "{web+page}&{hidden}&{hidden1}&{submit}");
...
...
```

Script example Form field as comments option not selected

The following example has the Form field as comments check box cleared.

```
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
...
...
for(i=0; i<2; i++)
Field[i]= NULL;
for(i=0; i<1; i++)
ActionURL[i]= NULL;
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/forms.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Forms Example", TITLE);
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1, &Field[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1, &Field[1]);
/* Request: 2 From: Forms Example */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
```

```
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name}&{password}&{e-mail}&{Address}&{city}&{state}"
        "&{zip}&{options}&{dates}&{comments}&{echo}&{test}"
        "&{web+page}&{hidden}&{hidden1}&{submit}");

...
...

for(i=0; i<2; i++)
{
free(Field[i]);
Field[i]=NULL;
}

for(i=0; i<1; i++)
{
free(ActionURL[i]);
ActionURL[i]=NULL;
}

...
...
```

Refresh timeout

Refresh timeout

When this option is selected, the time value that you specify in the seconds field is compared to a Web page's META Refresh value (e.g. <META HTTP-EQUIV=Refresh CONTENT="10"; URL="http://www.compuware.com">). If the META Refresh tag's CONTENT field value is less than the time value you specify, the page is treated as a redirected page. If the CONTENT field value is greater than the time you specify, the page is treated as a regular page.

This option is useful for avoiding infinite loops in the script. Infinite loops can occur if a page refreshes periodically to update data.

Example Web page
Script example with option selected
Script example with option not selected

Example Web page

```
<html>
<head>
<title>Just Wait</title>
<meta http_equiv=refresh content="5;url=/path/to/realpage.pl">
</head>
<body>
<h2>Loading the real page</h2>
</body>
</html>
```

Script example with the Refresh Timeout option selected

The following example has the Refresh Timeout check box selected and is set to a value greater than 5.

```
...
...
DO_SetTransactionStart();
DO_SetRefreshTimeout(10);
DO_SetMaxBrowserThreads(2);
...
...
/* Request: 1 */
DO_Http("GET http://host/path/to/page.pl HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("You have reached the final page!!", TITLE);
```

Script example with Refresh Timeout option not selected

The following example has the Refresh Timeout check box cleared. The example also applies to having the option selected and set to a value less than 5.

```
...
...
BEGIN_TRANSACTION();
DO_SetTransactionStart();
DO_SetMaxBrowserThreads(2);
...
...
/* Request: 1 */
DO_Http("GET http://host/path/to/page.pl HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Just Wait", TITLE);
DO_SLEEP(5);
/* Request: 2 */
DO_Http("GET http://host/path/to/realpage.pl HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("You have reached the final page!!", TITLE);
```

Streaming media

Streaming media

QALoad supports two types of streaming media:

- RealOne Player
- Windows Media Player

When streaming media conversion is enabled and you record a transaction that calls streaming media, an additional command is inserted into the script that requests the media. You do not have to listen to or view the entire media you are requesting; just record its URL and ensure that the appropriate media player is installed on the QALoad Player machines that will execute playback of the script. At run time, the script invokes the media player and requests the streaming media resource.

Note: Streaming media is not supported through firewalls and across proxies.

Advanced RealOne Player media options
Script examples with option selected
Script examples with option not selected

Advanced RealOne Player media options

The following statements are examples of how additional RealOne Player streaming media commands can be used in a script.

- `void ShowMediaRP(BOOL displayAudio, BOOL displayVideo);`

Enable/disable client audio and video for RealNetworks Streaming Media.

- `void EnableStatisticsRP(int statisticFlags, int interval, BOOL traceOutput);`

  Enable client-side performance statistics for RealNetworks Streaming Media.

- `void DisableStatisticsRP(void);`

  Disable client side performance statistics for RealNetworks Streaming Media.

Script examples with Streaming Media option selected

The following example has the Streaming Media check box selected and uses RealOne Player.

```
...
...
#include "do_www.h"
#include "RPLayer.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DownloadMediaRP("http://rm.host.com:8099/ramgen/demo.rm", 0);
...
...
END_TRANSACTION();
...
...
```

The following example has the Streaming Media check box selected and uses Windows Media Player.

```
...
...
#include "do_www.h"
#include "SMPLayer.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_Http("GET http://www.host.com/wmp-test.asx HTTP/1.0\r\n\r\n");
DownloadMediaFromASX(0);
...
...
END_TRANSACTION();
...
...
```

Script examples with Streaming Media option not selected

The following example has the Streaming Media check box cleared and uses RealOne Player.

```
...
...
#include "do_www.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_Http("GET http://rm.host.com:8099/ramgen/demo.rm HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

The following example has the Streaming Media check box cleared and uses Windows Media Player.

```
...
...
#include "do_www.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle(" QALoad WWW Capture Examples", TITLE);
DO_Http("GET http://www.host.com/wmp-test.asx HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

ActiveData

ActiveData

When this option is enabled, the generated script will automatically be variablized. Otherwise, all requests are not modified after the capture. The following example shows a capture of a form, how it is variablized by the ActiveData option, and not variablized when the ActiveData option is cleared.

Example Web page
Script example with option selected
Script example with option not selected

Example Web page

```
<!DOCTYPE HTML PUBLIC "-//AdvaSoft//DTD HTML 3.2 extended 961018//EN">
<HTML>
<HEAD>
<TITLE>Forms Example</TITLE>
</HEAD>

<BODY BGCOLOR="#7093DB">

<H2 ALIGN=center>Example of HTTP Forms</H2>
<BR>

<FORM ACTION="/cgi-bin/perl_9.pl" method=post>
<TABLE>
<TR>
<TD>Name:
<TD><INPUT NAME="name" SIZE="20" MAXLENGTH=20>

<TR>
<TD>Password:
<TD><INPUT TYPE =password NAME="password" SIZE="20" MAXLENGTH=20>

<TR>
<TD>E-Mail Address:
<TD><INPUT NAME= "e-mail" SIZE = "40" MAXLENGTH=80>

<TR>
<TD>Address:
<TD><INPUT TYPE = text NAME = "Address" SIZE = "40" MAXLENGTH=40>

<TR>
<TD>City:
<TD><INPUT NAME="city" SIZE="40" MAXLENGTH=40>
<TD ALIGN=left>State:
<TD ALIGN=left><INPUT NAME="state" SIZE="2" MAXLENGTH=2 ALIGN=left>
<TD ALIGN=left>Zip:
<TD ALIGN=left><INPUT NAME="zip" SIZE="5" MAXLENGTH=5>

<TR>
<TD VALIGN=top>Favorite Color:
<TD><SELECT NAME="options">
<OPTION>Red
<OPTION>Orange
<OPTION>Yellow
<OPTION>Green
<OPTION selected=on>Blue
<OPTION>Indigo</OPTION>
<OPTION>Violet</OPTION>
</SELECT>

<TR>

<TR>
<TD VALIGN=top>Color of your money:
<TD><SELECT NAME="dates" multiple="multiple">
<OPTION selected=on>Red
<OPTION>Blue
<OPTION>Green</OPTION>
<OPTION>Beige</OPTION>
</SELECT>

<TR>
<TD VALIGN=top>Comments:
<TD><TEXTAREA NAME="comments" COLS=40 ROWS=5></TEXTAREA>
</TABLE>

<BR><INPUT TYPE=checkbox CHECKED NAME="echo">Echo a copy of the result HTML Page to E-
```

```
mail<BR><BR>
<P>
<TABLE>

<TR>
<TD VALIGN=top>Testing:
<TD><INPUT TYPE=radio CHECKED NAME="test" VALUE="capture">Capture<BR>
<INPUT TYPE=radio NAME="test" VALUE="replay">Replay<BR>
<INPUT TYPE=radio NAME="test" VALUE="loadtest">Loadtest<BR>

<TR>

<TR>
<TD>Web page to append to reply:
<TD><INPUT TYPE=file NAME="web page">
</TABLE>

<BR>
There is a hidden field containing data here: <INPUT TYPE=hidden NAME="hidden" VALUE="This
rocks!">
Here is another hidden field: <INPUT TYPE=hidden NAME="hidden1" VALUE="Web testing is fun">
<BR>
<BR>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="colors.gif" width="200" height="100">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="eye.gif" width="80" height="60">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="devplatform.gif" width="48" height="43">
</TABLE>

<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This

<TR>
<TD><INPUT TYPE=image SRC="enterprise_sm.gif" width="42" height="41">
</TABLE>

<BR>

<TABLE ALIGN=center>
<TR>
<TD><INPUT TYPE=submit NAME="submit">
<TD><INPUT TYPE=reset>
</TABLE>
</FORM>
```

```
</BODY>
</HTML>
```

Script example with the ActiveData option selected

When this option is enabled, variablizations are done in the script based on the settings of other conversion options that become available. The following example has the ActiveData check box selected.

```
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
...
...
for(i=0; i<2; i++)
Field[i]= NULL;

for(i=0; i<1; i++)
ActionURL[i]= NULL;
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/forms.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Forms Example", TITLE);
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1, &Field[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1, &Field[1]);
/* Request: 2 From: Forms Example */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name}&{password}&{e-mail}&{Address}&{city}&{state}&"
        "{zip}&{options}&{dates}&{comments}&{echo}&{test}&"
        "{web+page}&{hidden}&{hidden1}&{submit}");
...
...
for(i=0; i<2; i++)
{
free(Field[i]);
Field[i]= NULL;
}
```

```
for(i=0; i<1; i++)
{
free(ActionURL[i]);
ActionURL[i]=NULL;
}
...
...
```

Script example with the ActiveData option not selected

When this option is not selected, variablizations are not done in the script regardless of the other options. The following example has the ActiveData check box cleared.

```
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/forms.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Forms Example", TITLE);
DO_Http("POST http://www.host.com/cgi-bin/perl_9.pl HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "name=joe&password=&e-mail=&Address=&city=&state=&zip=&"
        "options=Blue&dates=Red&comments=&echo=on&test=capture&"
        "web+page=&hidden=This+rocks%21&"
        "hidden1=Web+testing+is+fun&submit=Submit+Query");
DO_VerifyDocTitle("Forms Example - Results", TITLE);
...
...
END_TRANSACTION();
...
...
```

Testing a Script

Script Validation

Before adding a script to a load test, validate it to ensure that it runs without problems. The following procedure is only valid for Win32 scripts. To validate a UNIX script, see Validating a UNIX script.

To configure the Script Development Workbench and Player for validation:

1. Click **Options>Workbench** and select the **Script Validation** tab.
2. Select the **Automatically Recompile** check box if you want *QA*Load to compile a script before attempting to validate it. *QA*Load lists any compilation errors in the editor after compiling.
3. (For Java and OFS) select **Ask for Automatic Validation of Java and OFS Scripts**.
4. Select the **Only Display Player Output on Script Failure** check box to view only Player messages upon script failure.
5. Type a value in the **Wait up to** field. This is the number of seconds that the *QA*Load Script Development Workbench should wait for a script to execute before timing out.
6. In the Player Settings area, select the **Abort on Error** check box for *QA*Load to stop script execution upon encountering an error.

7. Select the **Debug Data** check box for the script to display a debug message indicating which command the script is executing.
8. In the Run As area, indicate whether the transaction should be run as **thread**- or **process-based**.

Note: Oracle Forms Server, Citrix, Java, and Uniface scripts are limited to process-based validation only.

9. In the **Number of users** field, type a number of virtual users to run this script for validation. The default is 1.
10. Enter a value in the **Transactions** field. For validation, Compuware recommends that you accept the default value of 1 transaction.
11. In the **Sleep Factor %** field, type the percentage of each DO_SLEEP (pause in the script) to maintain. For validation, you may not need to run every pause in the script at its full length. The value can be a percentage between 0 and 100. The default is 0.
12. Click **OK** to save your changes.

To validate the script:

1. In the Workspace Pane, click the **Scripts** tab.
2. Double-click on the appropriate script name to open the script.
3. From the **Session** menu, choose **Validate Script**.

You receive a message and Trace information in the Output pane. When the script executes successfully, you receive a confirmation message. If it does not execute successfully, use the Trace information to help you identify errors.

Debugging a Script

Log files

Log files can be generated for Oracle, Oracle Forms Server, Citrix, WWW, Uniface, ODBC, SAP, and Winsock scripts only.

If you encountered errors while validating or testing a script, you can view any log files generated during the test from the Script Development Workbench's LogFiles tab. Log files are generated during a test if you set debug options while setting up your test in the Conductor. Each virtual user for which you enabled Logfile Generation will have created a file containing information about its performance. When a test finishes running, all log files are saved in the directory `\Program Files\Compuware\QALoad\LogFiles`. Log files are named <scriptname>_<middleware>_vu<AbsoluteVirtualUserNumber>.<ext>, where:

! *<scriptname>* is the name of the script the virtual user ran

! *<middleware>* is the name of your middleware application

! *<AbsoluteVirtualUserNumber>* is the identification number assigned to the virtual user

! *<.ext>* is the file extension, dependent upon which middleware application you are testing. File extensions are listed in the following table:

Middleware	File Extension
Oracle WWW Citrix	.rip — A log file generated by a failed Player. At the end of a test, all .rip files are sent from the Players to the `\QALoad\LogFiles` directory and added to the merged timing file for your analysis.
Uniface WWW	.cap — A standard log file containing information about all statements executed during a test.
Citrix	.log — A standard log file containing information about all

ODBC Oracle Forms Server SAP Winsock WWW	statements executed during a test.

Verifying script checkpoints

You can quickly verify the syntax of the checkpoint commands BeginCheckpoint() and EndCheckpoint() in your script every time you compile your script by setting a single option, or on-the-fly with a single menu command.

To automatically verify script checkpoints, every time you compile a script:

1. From the Script Development Workbench's main menu, click **Options>Workbench**.
2. On the Configure Script Development Workbench dialog box, click the **Compiler Settings** tab.
3. Select the **Verify Checkpoints** option.
4. Click **OK**.

Every time you compile your script, the Script Development Workbench verifies the syntax of your checkpoint statements, and ensures the parameters passed in each pair match. If any errors are encountered, an error message displays in the Output pane. You can click on any error line to go directly to that line in the script.

To manually verify script checkpoints, for the open script only:

With your script open in the Workbook pane, click Session>Verify Checkpoints.

The Script Development Workbench verifies the syntax of your checkpoint statements, and ensures the parameters passed in each pair match. If any errors are encountered, an error message displays in the Output pane. You can click on any error line to go directly to that line in the script.

Using EasyScript

ADO

ADO Recording Options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you select the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

Note: If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the Capture Initialization Phase check box in the Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box.

Automatic: Select this option for QALoad to automatically start your application for recording, allowing you to record early application startup activity. This is the recommended method of recording calls, because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. Choose this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

Command Line: If you chose Automatic Program Startup, enter the command line of your application. You can also use the browse button to locate your application.

Working Directory: Enter the working directory of your application.

ADO Conversion Options

Field Retrieval: Select this option to include all instances of the ADO CARecordset->GetFields (represented in a QALoad script as ADO_Recordset(#)->GetFields(ADOFieldSet[#]); in the converted script.

Clearing this field removes a number of different elements from the script that may not be necessary for playback because they are processed on the client side rather than the server side. Removing them can greatly decrease the size of your script without affecting your load test results.

> Caution: Clear this option only if you are certain that ADO_Recordset(#)->GetFields(ADOFieldSet[#]); is not integral to your script.

For more information about this option, see Using the Field Retrieval option.

ADO Method Reference

QALoad provides descriptions and examples of the various methods that are available for an ADO script. For details, refer to the Language Reference Help section for ADO.

Citrix

Overview

Use QALoad's Citrix middleware to load test systems that run Citrix MetaFrame or Citrix MetaFrame XP.

What is Citrix?

Citrix middleware is a communication layer that provides remote access to Windows systems. The remote system appears in a window on the local system.

Connecting to the remote system

Once you have connected to a machine that is running the MetaFrame server, you can log in to the remote system and then run applications. Alternatively, you can specify an application in addition to a user name and password, which provides access only to the specified application and minimizes user input that is necessary to access the application under test.

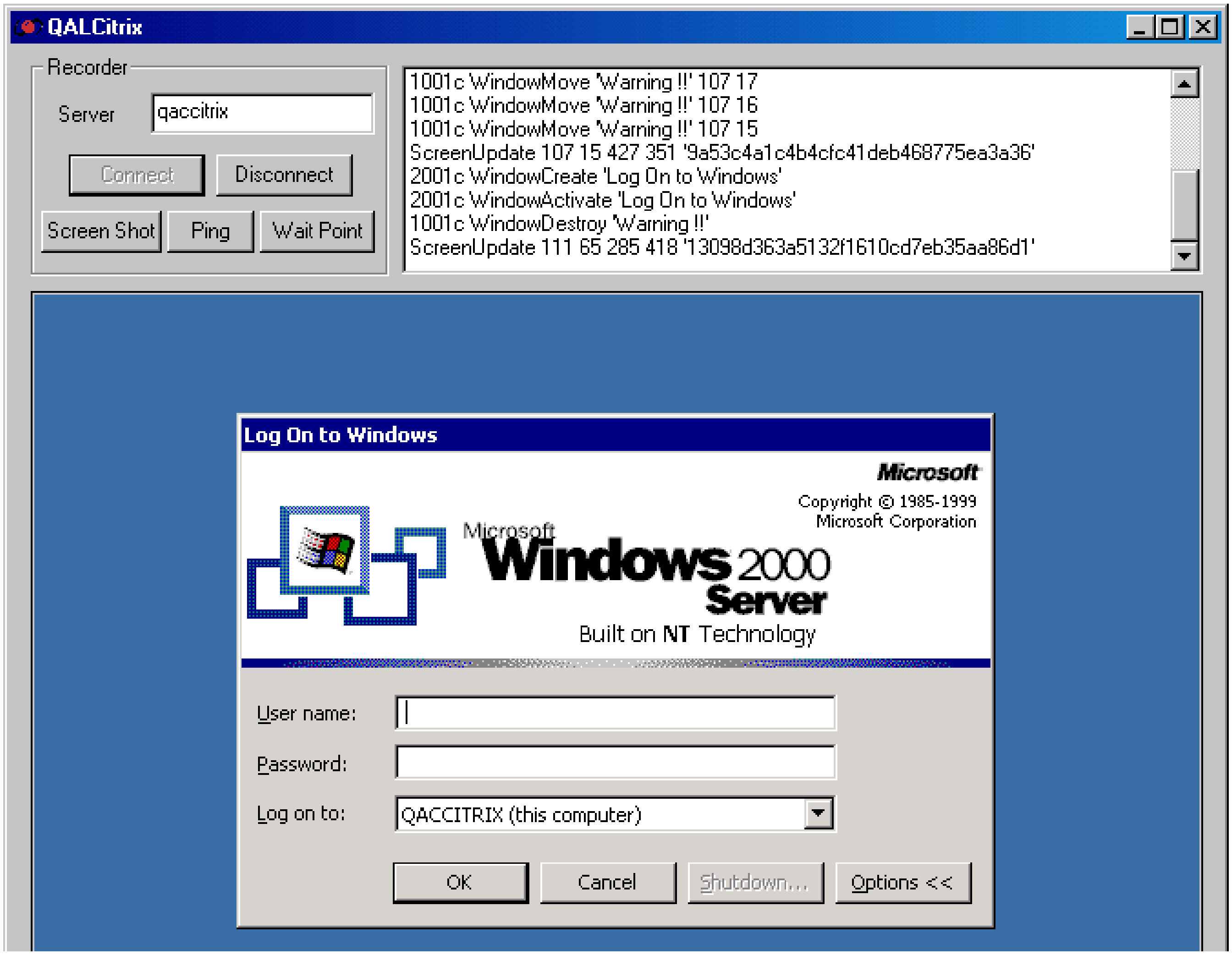
Testing in load-balanced environments

If you are testing an environment that includes a server farm, you can use Citrix ICA files to support this type of configuration. Specify the ICA file on the Citrix Record Options dialog box. ICA files are also necessary for encryption, Published Applications, and Published Desktops. ICA files are generated on the MetaFrame server and can be obtained from your MetaFrame administrator. For more information about using ICA files, see Using ICA files.

Recording a Citrix session

To begin recording a Citrix session, click the Record button on the Session toolbar. (If you have not already chosen Citrix as the session type, click the Citrix Session button to activate a new Citrix session.) The Citrix capture application appears, as shown in the following image.

Click the three sections of the image to learn more about the fields and the information that is displayed in each area.



Citrix recording options

To set recording options, choose Record... from the Options menu in the Script Development Workbench.

Set the following options for recording Citrix applications:

Server Information

Server: Type the name or address of the server machine for automatic connections. To connect to a server manually, do not enter a value in this field.

Username: Type the user name for the server machine that was specified in the Server field.

Password: Type the password for the user name that was specified in the Username field.

Domain: Type a domain name that applies to the user name and password that were specified in the Username and Password fields. Specifying a domain is optional. To ensure that the specified user name is logged on to the server instead of a domain, type the server name in this field.

Application Information

Select whether to automatically start an application after logging on. Choose Autostart to launch the application specified in the Application field, Custom to refer to the ICA file for application information, or None to disable automatic application startup.

Application: Type the path and file name of an application to start upon a successful log on.

Directory: Type the working directory for the application that was specified in the Application field. Specifying a working directory is optional even if you have specified an application in the Application field.

ICA File: Type the name of a Citrix ICA file. To enable this field, click the Custom option. Specify a URL or a file name without a path. This file, which is located on the MetaFrame server, contains configuration options for the Citrix client. For more information about ICA files, consult your Citrix administrator.

Notes: To test Published Applications and load-balanced environments (server farms), you must specify an ICA file.

To validate a script on the same machine on which it was captured, you must copy the ICA file to the `QALoad \BinaryFiles` directory. To use the ICA file on remote Player machines, the ICA file should be specified as an attached file in the External Data column of the Script Assignment tab in the Conductor.

Connection Information

Port: Type the port number over which Citrix ICA traffic travels. This port is also the port on which the MetaFrame server is listening. The default port is 1494.

Resolution: Choose a window size for the Citrix connection. To ensure that the entire Citrix interface is visible during recording, set the Resolution field in the Citrix recording options to a lower value than that of the desktop. Also, the screen resolution must be the same as the screen resolution specified in the Citrix ICA file.

Gracefully disconnect session: Select to specify that a logoff is issued before issuing a disconnect. If you do not select this option, you may need to configure the Citrix server to properly handle clean-up sessions.

Using ICA files

ICA files, which are generated on the MetaFrame server, contain configuration options for Citrix. You can specify an ICA file on the Citrix Record Options dialog box.

ICA files are specified in the script with the CtxSetICAFile command. If an ICA file is specified, the call is generated with an unqualified file name. For example:

```
CtxSetICAFile("customapp.ica");
```

Note: The file name is not fully-qualified because the file may not exist in the same location among the remote Player machines.

To validate the script on the same machine on which it was captured, copy the ICA file to the `QALoad\BinaryFiles` directory.

To use the ICA file on remote Player machines, the ICA file should be specified as an attached file in the External Data column of the Script Assignment tab in the Conductor.

Citrix conversion options

To set conversion options, choose Convert... from the Options menu in the Script Development Workbench.

Set the following options for Citrix conversions:

General

Replay Output Mode: Choose the playback mode to use during replay.

- Normal mode is normal headless replay.
- Renderless mode maximizes the number of possible client sessions (and minimizes CPU usage) by discarding all graphic data immediately after receipt. However, you cannot take snapshots in renderless mode.
- Windowless mode reduces CPU usage by allowing the client to skip drawing the screen image. Rendering still exists off-screen, which makes session snapshots possible.

Enable Counters: Select to enable the middleware counters that are built in to Connect(), Disconnect(), and Ping() calls. Enabling these counters can affect load test performance.

Keyboard/Mouse Input

Combine consecutive key characters into a string: Select to combine consecutive ASCII character key actions. This option combines individual calls to type characters into one call for the entire string.

Convert consecutive mouse commands into points/clicks: Select to consolidate consecutive MouseMove commands into a Point() command and to convert matching MouseDown/MouseUp command pairs into Click() commands.

Timeout Values

Connect: Type the number of seconds to wait for the connection to complete.

Ping: Type the number of seconds to wait for the Ping() call to return results.

Disconnect: Type the number of seconds to wait for the disconnection to complete.

Wait Point: Type the number of seconds to wait for wait points to complete.

Window events: Type the number of seconds to wait for window creation and destruction events to occur.

Restore Defaults: Click to set all timeout options to the default settings.

Window

Enable verification: Select to enable window verification. When this option is enabled, a window must be active for an action to be issued. Also choose the number of times to re-try for verification if an active window is not found, and the number of milliseconds to wait between each try.

Enable Wildcard Title Match: Select to enable wildcard comparisons for matching Citrix window creation events. For more information about wildcard comparisons, see Handling dynamic window titles.

Citrix command reference

QALoad provides descriptions and examples of the various commands available for a Citrix script. For details, refer to the Language Reference Help section for Citrix.

Advanced Scripting Techniques for Citrix

Handling Citrix server farms

Citrix servers can be grouped in farms. When load testing, you may want to connect to a Citrix server farm rather than to a specific server. This type of setup load tests the server farm and Citrix load balancing rather than a single server, which provides a more realistic load test.

To record a script that connects to a farm, you must use an ICA file to connect. However, when a capture takes place, a specific server (in the farm) must have a connection. Specify the correct ICA file to connect to the server farm as well as a specific server within that server farm. To verify that your script is connecting to a server farm and not a specific server, assign the server name to one blank space when validating the script. For example:

```
.
.
.

/* Declare Variables */
const char *CitrixServer   = " ";
const char *CitrixUsername = "citrix";
const char *CitrixPassword = "~encr~657E06726F697206";
const char *CitrixDomain   = "qacitrix2";
const int CitrixOutputMode = OUTPUT_MODE_NORMAL;

.
.
.

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("Orders.cpp");

CitrixInit(4);

/* Citrix replay settings */

CtxSetConnectTimeout(90);
CtxSetDisconnectTimeout(90);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, Citrix-Domain);
CtxSetICAFile("PRD desktop.ica");
CtxSetEnableCounters(TRUE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
```

Handling dynamic window titles

Some applications create windows whose titles vary depending on the state of the window. For example, Microsoft Word creates a title based on the default document name at the time of the window creation. During replay, this dynamic title can differ from the window title that was recorded, and the window is not recognized. If this occurs, try the following steps to modify the script:

1. **Ensure that the Enable Wildcard Title Match check box is selected in the Citrix conversion options prior to converting the recording.**
   In the Window Verification group of the **Citrix Convert Options** dialog box, ensure that the **Enable Wildcard Title Match** check box is selected. This check box is selected by default. If you are working with a previously-converted script, ensure that a CtxSetEnableWildcardMatching command exists in the script prior to the BEGIN_TRANSACTION command and that the parameter is set to TRUE.

2. **Verify whether there is an issue with dynamic window titles.**
   When a script fails on validation because the run time window title is different than the expected window title from the recording, it is likely that you are dealing with a dynamic title issue that can be handled by this scripting technique. In this case, the script fails on the CtxWaitForWindowCreate call.

3. **Identify a match “pattern” for the dynamic window title.**
   Note the error message that is returned during validation (or replay). The message indicates the expected window title versus the window title from script playback. Examine the differences in the window titles to create a “match pattern” that recognizes the window title, while ignoring other windows. A match pattern can be a simple substring of the window title or a pattern string using wildcard characters such as ? (to match any single character) or * (to match any number of characters). The examples below illustrate the different match patterns.
4. **Insert a CtxSetWindowMatchTitle command prior to the CtxWaitForWindowCreate call for the dynamic window.**
   When adding the SetWindowMatchTitle command, ensure that the first parameter contains the correct window object and the second parameter contains the match string in double-quotes.
5. **Validate the script to ensure the CtxWaitForWindowCreate command recognizes the dynamic window name.**
   Run the revised script through validation to ensure that the script succeeds. If the script does not validate successfully, go to step 3 to determine if the match pattern is correct.

Example 1: Using a substring match

In this example, the Microsoft Word application generates a dynamic title when the script is replayed. The dynamic name is a concatenation of the default document that Word creates at application startup with the name of the application. The script is altered to reflect the fact that the string “Microsoft Word” is always part of the window title:

```
// Window CWI_13 ("Microsoft Word") created
CtxSetWindowMatchTitle( CWI_13, "Microsoft Word" );
CtxWaitForWindowCreate(CWI_13);
```

Example 2: Using a wildcard match with the * character

In this example, the SampleClientApp application generates a dynamic title when the script is replayed. The dynamic name is the name of the application followed by the name of the user, beginning with the word “User”. The asterisk (*) wildcard is substituted for a given username, reflecting the pattern of “SampleClientApp – User:” as part of the window title followed by an arbitrary user name:

```
// Window CWI_13 ("SampleClientApp - User: John") created
CtxSetWindowMatchTitle(CWI_13,"SampleClientApp – User: *" );
CtxWaitForWindowCreate(CWI_13);
```

Example 3: Using a wildcard match with the ? character

In this example, the RandomValue application generates a dynamic title when the script is replayed. The dynamic name is the application followed by a random single digit. The question mark character is substituted for the single digit to reflect the pattern that begins “RandomValue: ”, followed by single digit:

```
// Window CWI_13 ("RandomValue: 0") created
CtxSetWindowMatchTitle( CWI_13, "Sample Application: ?" );
CtxWaitForWindowCreate(CWI_13);
```

Handling dynamic windows

During conversion, CtxWaitForWindowCreate calls are added to the script for each named window creation event. During replay, some dynamic windows that were in the capture may not appear, which causes the script to fail because a wait point times out. To avoid script failure in this circumstance, comment out the CtxWaitForWindowCreate commands that may be referencing dynamic windows.

Handling dynamic windows that require user interaction

Some windows that require user action before normal script processing can proceed may appear intermittently during replay. One example commonly encountered with Citrix is the ICA Seamless Host Agent window. This window, if it appears, requires user action or the script may fail.

To work around this issue, follow these steps:

1. Capture a session in which the dynamic window appears and the user performs the action to dismiss the window. This may require multiple attempts to capture the window. Once this is captured in a recording, save the script as a temporary script.
2. If the window did not appear in the primary script, extract the code snippet from the temporary script that acts on the dynamic window and insert it into the real script. The code usually consists of a CtxPoint command and a CtxClick command for this window. Insert the commands after the CtxWaitForWindowCreate command for the dynamic window. In addition, extract and insert the Citrix window information object constructor call and delete call to the relevant parts of the script, changing the object name to avoid conflicting with existing window objects. Ensure that the additional code is not inserted between a CtxPoint command and a CtxClick command in the primary script.
3. Add a special CtxSetWindowMatchTitle command immediately before the CtxWaitForWindowCreate command. The first parameter of the CtxSetWindowMatchTitle command should be the correct window object. The second parameter contains a special wildcard match "*" that enables the CtxClick command to accept any window title, which ensures that even if the matching window does not appear, the command still executes successfully.
4. If the window appears in the primary script, comment out the CtxWaitForWindowCreate command for the dynamic window. Because the window itself may not appear, the CtxWaitForWindowCreate command should be commented out.
5. Validate the script. If the validation is not successful, ensure that steps 2-4 were performed correctly.

In the following example's scenario, the ICA Seamless Window Agent window does not appear in the primary script, but appears intermittently when the primary script is replayed, causing those replay sessions to fail. A second Citrix script, which includes the window appearance, is recorded and the CtxPoint and CtxClick commands are extracted from this script and inserted into the primary script, with the window object changed to match the object in the primary script. In addition, the Citrix window object constructor call and delete call are added in the appropriate places in the script, and the CtxClick command is changed to refer to this object. In the following example, the text in bold represents code that was manually inserted into the location in the primary script where the window appears in the secondary script.

```
CtxWI *CWI_99 = new CtxWI(0x10052, "ICA Seamless Host Agent", 0, 0, 391, 224);
...
CtxSetWindowMatchTitle( CWI_99, "*" );
CtxPoint(190, 203);
CtxClick(CWI_99, 0, L_BUTTON, NONE);
CtxPoint(300, 400);
...
delete CWI_99; // "ICA Seamless Host Agent"
```

Handling unexpected events in Citrix

The CtxWindowEventExists and CtxScreenEventExists commands can be used to handle unexpected window and screen events in Citrix scripts. When there is a possibility of unexpected dialogs appearing or unexpected screen events occurring, you must modify the script to respond to the changes and continue the load test.

For example, if a script opens a Microsoft Word document that resides on a network, and that document is already open by another network user, an unexpected dialog box appears that prompts the user to choose between continuing to open the document in read-only mode or to cancel it. To prevent script failure, modifications can be made in the script to handle the dialog boxes that appear in this situation.

Generally, to handle unexpected events, you record two scripts. The first script contains a recording of the expected events. The second script should include the unexpected events. Using the CtxWindowEventExists and CtxScreenEventExists functions, create a conditional block of code that handles the dialogs that may appear.

Example

The following script example shows the additional script lines that were added to handle a Word document that is already open by another user on a network. The added lines appear in boldface type.

```
/*
 *  capSave11111-2.cpp
 *
 *  Script Converted on June 21, 2004 at 01:04:17 PM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 50
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */

/* Converted using the following options:
 * General:
 *  Line Split                          : 132 characters
 *  Sleep Seconds                       : 1
 *  Auto Checkpoints                    : No
 * Citrix
 *  General Options                     :
 *   Window Verification                : Yes
 *   Session Timeouts                   : Yes
 *    Connect Timeout (s)               : 60
 *    Disconnect Timeout (s)            : 60
 *    Window Creation Timeout (s)       : 30
 *    Ping Timeout (s)                  : 20
 *    Wait Point Timeout (s)            : 30
 *   Include Wait Points                : Yes
 *   Enable Counters                    : No
 *   Include Unnamed Windows            : Yes
 *   Output Mode                        : Normal
 *  Input Options                       :
 *   Combine Keyboard Input             : Yes
 *   Combine Mouse Input                : Yes
 */

#define CITRIX_CLIENT_VERSION "8.00.60000"
#define CITRIX_ICO_VERSION    "2.4"
#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
            /* Declare Variables */
            const char *CitrixServer      = "qaccitrix";
            const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

            /* Citrix Window Information Objects */
            CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
            CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
            CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
            CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198,
397, 127);
            CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198,
```

```
397, 127);
            CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
            CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
            CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
            CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
            CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
            CtxWI *CWI_11 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
            CtxWI *CWI_12 = new CtxWI(0x2006a, "", 200, 186, 156, 287);
            CtxWI *CWI_13 = new CtxWI(0x10138, "", 112, 116, 416, 248);
            CtxWI *CWI_14 = new CtxWI(0x50036, "Microsoft Word", -4, -4, 649, 461);
            CtxWI *CWI_15 = new CtxWI(0x1017e, "Open", 19, 23, 602, 387);
            CtxWI *CWI_16 = new CtxWI(0x20174, "*Microsoft Word", -4, -4, 649, 461);
            CtxWI *CWI_17 = new CtxWI(0x10058, "", 113, 114, 305, 26);
            CtxWI *CWI_18 = new CtxWI(0x2013e, "Calculator", 66, 66, 261, 253);
            CtxWI *CWI_19 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
            CtxWI *CWI_20 = new CtxWI(0x3006a, "Shut Down Windows", 111, 96, 418, 193);

            CtxWI *CWI_117 = new CtxWI(0x20172, "File In Use", 144, 127, 352, 179);
            CtxWI *CWI_118 = new CtxWI(0x30172, "11111111 (Read-Only) - Microsoft Word", -4,
-4, 649, 461);

            SET_ABORT_FUNCTION(abort_function);

            DEFINE_TRANS_TYPE("capSave11111-2.cpp");

            CitrixInit(1);

            /* Citrix replay settings */
            CtxSetConnectTimeout(60);
            CtxSetDisconnectTimeout(60);
            CtxSetWindowTimeout(30);
            CtxSetPingTimeout(20);
            CtxSetWaitPointTimeout(30);
            CtxSetWindowVerification(TRUE);
            CtxSetEnableCounters(FALSE);
            CtxSetWindowRetries(5, 5000);
            CtxSetEnableWildcardMatching(TRUE);

            SYNCHRONIZE();

            BEGIN_TRANSACTION();

            DO_SetTransactionStart();

            CtxConnect(CitrixServer, CitrixOutputMode);

            // Window CWI_1 ("Warning !!") created 1087837356.454

            CtxWaitForWindowCreate(CWI_1, 2125);

            DO_MSLEEP(1891);
            CtxPoint(246, 267);  //1087837358.797

            DO_MSLEEP(453);
            CtxMouseDown(CWI_1, L_BUTTON, NONE, 246, 267);  // 1087837358.797

            CtxMouseUp(CWI_1, L_BUTTON, NONE, 247, 267); //1087837359.032

            .
            .
            .

            DO_MSLEEP(63);
            // Window CWI_14 ("Microsoft Word") created 1087837397.390
```

```
            CtxWaitForWindowCreate(CWI_14, 141);

            DO_MSLEEP(78);
            CWI_14->setTitle("Document1 - Microsoft Word"); //1087837397.468

            // Window CWI_13 ("") destroyed 1087837397.468

            DO_MSLEEP(2468);
            CtxPoint(37, 50);  //1087837400.218

            DO_MSLEEP(282);
            CtxClick(CWI_14, 203, L_BUTTON, NONE); //1087837400.421

            // Window CWI_15 ("Open") created 1087837400.764

            CtxWaitForWindowCreate(CWI_15, 344);


            DO_MSLEEP(1656);
            CtxPoint(132, 99);  //1087837402.671

            DO_MSLEEP(250);
            CtxDoubleClick(CWI_15); // 1087837402.874

            DO_MSLEEP(109);


            DO_MSLEEP(1953);
            CtxPoint(247, 197);  //1087837404.827

            // Window CWI_15 ("Open") destroyed 1087837404.827


            if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE,3000,CWI_16))
            BeginBlock();
                        CtxPoint(337, 265);  //1087837404.905

                        // Window CWI_16 ("11111111 - Microsoft Word") created
1087837404.905

                        CtxWaitForWindowCreate(CWI_16, 31);

                        // Window CWI_14 ("Document1 - Microsoft Word") destroyed
1087837404.905

                        DO_MSLEEP(7547);
                        CtxPoint(628, 9);  //1087837414.592

                        DO_MSLEEP(2141);
                        CtxClick(CWI_16, 281, L_BUTTON, NONE); //1087837414.873

                        DO_MSLEEP(234);
                        // Window CWI_16 ("11111111 - Microsoft Word") destroyed
1087837415.108

                        CtxPoint(113, 93);  //1087837418.779

                        // Window CWI_17 ("") created 1087837418.779
            EndBlock()

///ReadOnly Code Start

            else
            BeginBlock();
```

```
                        // Window CWI_117 ("File In Use") created 1087840076.599

                        CtxWaitForWindowCreate(CWI_117, 578);

                        DO_MSLEEP(2360);
                        CtxPoint(358, 283);  //1087840079.068


                        DO_MSLEEP(125);
                        CtxClick(CWI_117, 281, L_BUTTON, NONE); //1087840079.365


                        DO_MSLEEP(109);
                        // Window CWI_117 ("File In Use") destroyed 1087840079.458

                        // Window CWI_118 ("11111111 (Read-Only) - Microsoft Word") created
1087840079.521

                        CtxWaitForWindowCreate(CWI_118, 63);

                        // Window CWI_115 ("Document1 - Microsoft Word") destroyed
1087840079.521

                        DO_MSLEEP(4766);
                        CtxPoint(631, 3);  //1087840084.490

                        DO_MSLEEP(203);
                        CtxClick(CWI_118, 250, L_BUTTON, NONE); //1087840084.740

                        DO_MSLEEP(93);
                        // Window CWI_118 ("11111111 (Read-Only) - Microsoft Word")
destroyed 1087840084.833

                        DO_MSLEEP(2407);
                        CtxPoint(34, 465);  //1087840087.333

            EndBlock();

///ReadOnly Code End


            DO_MSLEEP(1063);

            DO_MSLEEP(484);
            CtxPoint(112, 93);  //1087837419.654

            DO_MSLEEP(406);
            CtxDoubleClick(CWI_9); // 1087837419.904
            .
            .
            .

            // Window CWI_9 ("Program Manager") destroyed 1087837440.122

            // Window CWI_7 ("") destroyed 1087837440.138

            DO_SetTransactionCleanup();

            CtxDisconnect();


            END_TRANSACTION();

            delete CWI_1; // "Warning !!"
            delete CWI_2; // "Log On to Windows"
```

```
            delete CWI_3; // "Please wait..."
            delete CWI_4; // "Citrix License Warning Notice"
            delete CWI_5; // "Citrix License Warning Notice"
            delete CWI_6; // "UsrLogon.Cmd"
            delete CWI_7; // ""
            delete CWI_8; // "ICA Seamless Host Agent"
            delete CWI_9; // "Program Manager"
            delete CWI_10; // ""
            delete CWI_11; // ""
            delete CWI_12; // ""
            delete CWI_13; // ""
            delete CWI_14; // "Microsoft Word"
            delete CWI_15; // "Open"
            delete CWI_16; // "11111111 - Microsoft Word"
            delete CWI_17; // ""
            delete CWI_18; // "Calculator"
            delete CWI_19; // ""
            delete CWI_20; // "Shut Down Windows"

            delete CWI_117; // "File In Use"
            delete CWI_118; // "11111111 (Read-Only) - Microsoft Word"

            CitrixUninit();

            REPORT(SUCCESS);
            EXIT();
            return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
            RR__printf("Virtual User ABORTED.");

            CitrixUninit();

            EXIT();
}
```

Moving the Citrix connect and disconnect outside the transaction loop

If your load testing requirements for Citrix include creating extended logon sessions, in which the user remains connected to the Citrix server between transactions, review the following tips for recording and script development.

Recording

Perform the following steps during the recording process in order to prepare for moving the connect and disconnect actions outside the transaction loop:

1. Insert a comment such as “Logged in to Citrix” after the Citrix logon but before any windows have been opened.
2. Ensure that all application windows are closed before disconnecting from the Citrix session.
3. Insert a comment such as “Ready to log off Citrix” before the Citrix logoff sequence is initiated. Ensure that the first comment is added after the user has logged on and closed all login-related dialog boxes, but before any applications are started. Similarly, the second comment must be placed after all applications have been closed, but before the user logs off.

Scripting

Comment out the BEGIN_TRANSACTION and END_TRANSACTION calls and add new BEGIN_TRANSACTION and END_TRANSACTION calls at the location where the comments from steps 1 and 3 above were placed. Comment out the calls instead of deleting them so that the original location of these commands can be determined for debugging purposes.

Also comment out the DO_SetTransactionStart and DO_SetTransactionCleanup calls.

Using the CtxWaitForScreenUpdate command

In some situations, a window may vary in how long it takes to refresh on the screen. For example, the Windows Start menu is an unnamed window that can take varying amounts of time to appear, depending on system resource usage. To prevent playback problems in which a mouse click does not synchronize with its intended window, insert the CtxWaitForScreenUpdate command in the script after the action that causes the window to appear. The parameters for the CtxWaitForScreenUpdate command correspond to the X and Y coordinates and the width and height of the window. This command ensures that the window has enough time to display before the mouse click.

Java

Overview

QALoad does not support recording of Java scripts. Instead, Java scripts are created from script templates that you use to create a stub script that you can then edit manually. Templates are saved in the `QALoad\Middlewares\Java\Templates` directory. QALoad supplies four default templates. QALoad uses a token name to represent the classpath — when you create a new Java script, QALoad simply replaces the token `<classnamehere>` with the class/script name you assign. You can also install additional templates using the `<classnamehere>` token if you wish.

Following is an example of a Java template with the classname token (in bold):

```
import com.compuware.qacenter.qaload.EasyScript.*;
public class <classnamehere> implements EasyScript

     {
     /** optional - Class method runs once for each script when class is loaded */
     public static void setup QALoad Test () throws Exception
     {
     }
```

Use the Script Development Workbench to create a new EasyScript for Java script from the provided stub scripts.

Using my script from a previous version of QALoad

Accessing JavaDoc

QALoad provides JavaDoc for your reference. To access it from the Script Development Workbench menu, choose Help>EasyScript for Java: JavaDoc from a Java session.

Creating a Java Script

To create a Java script for QALoad :

1. With a Java session open, choose **File>New** from the menu.
2. In the File area, click on the **Middleware** tree item.

3. In the Filename field, type a name for your new Java script. Note that Java file names do have special requirements, and *QA*Load will enforce those requirements. For example, Java file names cannot contain spaces. If you try to include a space in your file name, *QA*Load will give you an error prompt.
4. Click **OK**. The Create Java Script dialog box opens.
5. Under the Script field is a selection box listing all the templates available in your `\QALoad\Middlewares\Java\Templates` directory. *QA*Load provides four default templates. If you click on a template name, a sample is shown in the right pane. The four templates are:
   - long format — Provides all required and optional methods.
   - new class — Creates a class associated with the script.
   - old format — Shows modifications needed to run legacy scripts.
   - short format — Provides only the minimum required methods.

   Select the template that best suits your needs and click OK. QALoad creates a stub script by the name you designated and opens it in the Workbook pane for editing.
6. Edit your script as necessary. You can use *QA*Load's Java Script Options dialog box to edit some script attributes.

Note: The main call in the script is used for debugging proposes and is not executed in the Conductor.

Oracle

Oracle recording options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

Note: If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the Capture Initialization Phase check box on the Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box.

Automatic: Select this option for QALoad to automatically start your application when recording, allowing you to record early application startup activity. This is the recommended method of capturing API calls, because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. Select this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

Command Line: If you chose Automatic Program Startup, enter the command line of your Oracle application. You can also use Browse to locate your application.

Working Directory: Enter the working directory of your Oracle application.

Note: If you entered the full path in the Command Line field, this field is filled in automatically.

Oracle conversion options

Database Paths, Includes: Enter the location of your Oracle database includes.

Database Paths, Libraries: Enter the location of your Oracle database libraries.

Variablization (ActiveData) Enable: Select this option to enable ActiveData for Oracle during conversion. This option is enabled by default. For more information, click ActiveData for Oracle.

Minimum Characters: Enter the minimum number of characters to match for auto-variablization.

PostCapture: Fetch Iteration Override: Type the number of fetch iterations allowed while recording a script to control the amount of data fetched during playback. To fetch all data, type: 1000000.

ActiveData for Oracle

ActiveData for Oracle

Oracle variablization is a powerful scripting assistant that provides automatic correlation of data values in your script (auto-variablization) and lets you use a datapool as the source of data values (manual-variablization).

Auto-variablization

When you enable auto-variablization, QALoad correlates the data values produced by the execution of recorded SQL statements and assigns a single source variable to matching bind and static variables that subsequently use the value. Auto-variablization will only target a capture file's bind variables and embedded static data in recorded SQL statements as receivers of source variables. Source variables will be automatically generated based on the capture file's PostBind data, Fetch data, and embedded Static data in SQL statements. Source variables from PostBind data will be generated only if the PostBind data belongs to one of these OCI bind data types:

Code	OCI7 Bind Data Type	OCI8 Bind Data Type
3	SQLT_INT	SQLT_INT
4	SQLT_FLT	SQLT_FLT
68	SQLT_UIN	SQLT_UIN
1	SQLT_CHR	SQLT_CHR
5	SQLT_STR	SQLT_STR
96	SQLT_AFC	SQLT_AFC
97	SQLT_AFC	SQLT_AFC
11	SQLT_RID	SQLT_RID Not Applicable in OCI8

Source variables from Fetch data will be generated only if the Fetch data belongs to one of the above OCI datatypes or one of the following:

Code	OCI7 Fetch Data Type	OCI8 Fetch Data Type
6	SQLT_VNU	SQLT_VNU
2	SQLT_NUM	SQLT_NUM

Note: Fetch data is made available in the capture file only when the Oracle Capture Option Use Fetch data for Variablization is selected.

Static data embedded in SQL statements will be used as source variable or receiver of a source variable only when the SQL statement states a SELECT, INSERT, UPDATE or DELETE operation. SQL statements that contain stored procedures (e.g. BEGIN…) will be excluded.

Auto-variablization occurs by default in QALoad, but you can turn it off by clearing the conversion option Variablization (ActiveData) on the Oracle Convert Options tab. If you choose to use automatic variablization, you can then use manual-variablization to change a source variable previously determined by auto-variablization to data from a local or central datapool.

Manual Variablization

Manual variablization allows you to change the source of variables identified through auto-variablization to use data from central or local datapools. You use the variablization tree-view and the options available from the tree-view to view and change source variables.

Manual variablization is limited to changing the source variables to data that was prepared from a local datapool or conductor (central) datapool. Once changed, all (but not individual) source variables may be changed back to the original source variables.

Why use ActiveData for Oracle?

- **To avoid duplicate key errors which can occur during playback when the data relationships hidden (implied) within a set of Oracle SQL statements are not recorded.** For example, a recorded Select SQL statement may include the Oracle nextval expression to get the next sequential unique number in the database. The returned value from the expression is used for the primary key in a subsequent Insert statement. The primary key is associated with a bind variable. The value of the bind variable is recorded and noted in the *QA*Load script. When the script is played back, the returned value from nextval will naturally be different from the value of the bind variable. The Insert SQL execution incurs a duplicate key error from the Oracle server.

  Oracle variablization prevents this error by providing a logical relationship between the returned data from the Select statement and the data for the Insert bind variable. The data relationship is established through a source variable.

- **To reduce diagnostic time for playback data issues, especially when dealing with large scripts.** Using a single source variable for script variables that have the same data value reduces the amount of debugging time that would have been spent on multiple script variables. Additionally, the Compare Tool aids you in debugging data issues by highlighting SQL and data differences that could influence the load test of two similar capture files.

Variablization menu

Access the Variablization menu from the Script Development Workbench's Session menu, or by right-clicking from the variablization tree-view.

Create/Edit a source: Opens a tree-view of your variablized statements and their sources.

Show Capture Difference: Accesses the Compare Tool, where you can choose a capture file to compare to the current capture file and have the differences in SQL statements and bind data highlighted for your comparison within the variablization tree-view.

Revariablize: Deletes all manually generated source variables and re-executes auto-variablization. Note that datapool sources may not be changed back to PostBind, Fetch, or Static data unless you select this option.

Remove all sources: Deletes all source variables from the script's .var file.

Show SQL statement: Provides a detailed view of the highlighted SQL statement. The detailed view will display associated Bind and Column data (from the Execute statement), associated PostBind data, and associated Fetch data.

Hints: Opens the Oracle Variablization Hints online help.

Word wrap: Shows the complete SQL statement in wrapped format. This is selected by default.

Display options: Allows you to change display options to one of the following: Only statements with bind variables, Unsourced Bind statements, or Show all SQL statements (default).

The Refresh the current view option will re-draw the tree-view after a source is manually changed.

Save the Variablization VAR file: Saves any changes to the script's .var file.

Save and Convert: Saves changes to the .var file and re-converts the script.

Save and Convert As: Saves changes to the .var file and prompts you to save your script under a new name before re-converting it.

Variablize

Use this dialog box to variablize a file or to compare two similar files. The results are displayed in a tree-view from which you can manually variablize the file or view the differences between the two files. When you compare two files, the differences in SQL statements and bind data are highlighted within the Variablization tree-view.

Variablize the following capture file: Lists the path and name of the currently selected capture file (.cap).

Compare and Variablize with the following file: Navigate to the capture file you'd like to compare to the currently selected capture file.

Variablize: Variablizes the file and displays the recorded SQL statements, bind variables, static variables embedded in SQL statements, data values and the sources of data values as determined by auto-variablization.

Cancel: Closes the dialog box without making any changes.

Source Details

Displays details about the source of the selected variable, and allows you to replace the source with data from a central or local datapool.

Name: Lists the name of the field in the script that was variablized.

Value: Lists the value assigned to the variablized field.

Line #: Lists the script line where the field is located.

(Default) From Postbind/Fetch/Static data: If this option is selected, the source of the variable was determined by auto-variablization.

Source variable name in Convert script: The name assigned to the variable by auto-variablization, or when replaced by a datapool variable.

From datapool: Select this option to change the source to a central or local datapool.

Field Number: Specify the column number in the datapool file to use as the source.

Advanced Options: Click to open the ActiveData Advanced Source Options dialog box where you can format the source before using it, if necessary.

Display values matched by auto-variablization: In this area, click the appropriate button to determine which values to display: Sources, Matching values, or Matching names and values.

Match exact: Select if the source must be an exact match, or de-select to use the source for a sub-string search.

Update Source: Click to update the variable source according to the settings on this dialog.

Update ALL: Click to use the newly created source variable for all items in the list area.

Delete Source: Click to delete the variable and all its references from the tree-view.

Quit: Click to cancel without saving any changes.

Comparing files

The Oracle Variablization Compare Tool compares two similar capture files and highlights the differences in SQL statements and bind data and highlights them in the Variablization Tree View.

Why use the Compare Tool?

The Compare Tool can help you debug data issues in your transaction that may cause load test problems, especially in large scripts. With the differences highlighted in a window display, you can quickly determine if manual variablization is warranted for specific variables. Manual variablization can help you work around data issues that influence load tests.

To use the Compare Tool:

1. In the Workspace pane, right-click on the first capture file you want to compare and select **Variablize** from the shortcut menu. The Variablize dialog box opens, displaying the path and name of the selected file.
2. Select the **Compare and Variablize with the following file** check box, and then navigate to the capture file you wish to compare against the first selected file.
3. Click the **Variablize** button. A new tab opens in the Script Development Workbench, presenting a tree-view of the data. Differences in SQL statements and bind data are highlighted.
4. View differing values by clicking on a highlighted bind variable or SQL statement. The Show Capture Difference window will open, listing the value used in each file.
5. If you don't need to change the data, click **OK** to be returned to the tree-view. If you need to change the source of a bind item to a datapool variable, click **Go to Source Display**. The Source Details (for bind data) window or Show SQL Statement (for SQL statements) window opens.
6. Change the source of any variables to call datapool items.
7. Save the .var file and convert your capture file to build an updated script by right-clicking and selecting **Save and Convert** or **Save and Convert As**.

Setting up QALoad to run Oracle scripts on UNIX

After installing the QALoad UNIX Player and utilities, you should ensure that the following environment variables are set prior to starting the Player Agent (loadagent):

Platform	Environment Variable	Value
All Platforms:	ORACLE_HOME	<path>/oracle/product/<version>
	TNS_ADMIN	<location of config files>
	ORACLE_SID	<oracle instance name
Linux:	LD_LIBRARY_PATH	<playerdir>/lib:<ORACLE_HOME>/lib

Solaris:	LD_LIBRARY_PATH	<playerdir>/lib:<ORACLE_HOME>/lib

Setting environment variables on UNIX systems depends on your login shell. For example:

- For ksh: `export ORACLE_HOME=/oracle/product/8.1.6`
- For csh: `setenv ORACLE_HOME /oracle/product/8.1.6`

The ORACLE_HOME environment variable points to the directory where the Oracle workstation software has been installed. The TNS_ADMIN environment variable should point to the location of the client and/or server config files. ORACLE_SID should be set to the name of the Oracle instance. For each UNIX platform, update the appropriate library path variable to include the library directory for the particular version of Oracle.

Scripts will automatically be downloaded to the Player machines by the Conductor and compiled, if necessary, at test execution time.

During the automatic script download and compile, if a script compile error occurs, a scriptname.err file will be generated in the scripts directory.

To compile a script by hand, use the Rmake command. The syntax is as follows:

```
Rmake <scriptdir>/<scriptname>
```

or

```
Rmake <scriptdir>/<scriptname>
```

Oracle command reference

QALoad provides descriptions and examples of the various commands available for an Oracle script. For details, refer to the Language Reference Help section for Oracle OCI Version 7, General Oracle, or Oracle OCI Version 8.

OFS

Recording from Oracle Forms Server

QALoad supports recording Oracle Forms 10g, 9i, and patched 6i (versions 6.0.8.14 and up) applications in HTTP mode (also called Servlet mode), Forms 6.0 and 6i applications in socket mode, and SSL-enabled Oracle Forms 9i and 10g applications. These recording types are described briefly below.

Recording Servlet Mode

Oracle 9iAS and Oracle Application Server 10g use HTTP to send Forms data across the network. To record in Servlet mode, select the appropriate Servlet Mode option in the Record Options dialog box before you start recording your application.

> Note: When using server-side recording, you must perform steps to configure the server. See Using server-side recording for more information.

Recording SSL Mode

To record an OFS Servlet Mode application in SSL mode, select the appropriate Servlet mode option in the Record Options dialog box, and enter the Jinitiator certDB file that the application uses. The certDB file verifies the SSL Certificate Authority on the client side prior to the Forms connection.

Note: SSL mode is not available with server-side 9i recording.

Recording Socket Mode

For Socket Mode recording, QALoad must start your application for you through your browser. Before recording, enter the URL of the Forms applet page in the URL field, and the Forms Server port in the Port field. If you leave the Port field blank, or enter an incorrect port number, your recording will only result in an empty capture file. You may leave the URL field blank, but will be prompted for the Forms applet page on the initial browser page. From the applet page, click the link to your Forms application. QALoad will take over recording at this point.

Using server-side recording

For Forms applications running in Oracle 9iAS, you can use server-side recording, which avoids issues that can be encountered with some server configurations. Server-side recording creates a script recording by using the Forms server's own capabilities to record transactions.

To enable server-side recording, select the 9i Server-Side Recording check box on the Oracle Forms Server Record Options dialog box and provide the URL of the ListenerServlet in the ListenerServlet field.

Server setup

Before recording a script using server-side recording, you must first modify the server configuration so that QALoad can communicate with the server properly. Once the server modifications are complete, recording and playback are the same as for standard OFS scripts.

To prepare the Forms server for server-side recording:

1. Copy the `ofsmessage.jar` file from the `\QALoad\Classes` directory of the *QA*Load installation on the client machine to the `\FORMS90\JAVA` directory of the Oracle 9i Application Server installation on the server machine.
2. Add a new section of configuration parameters to the `formsweb.cfg` file in the `\FORMS90\server` directory of the Oracle 9i Application Server installation. Use the following format, substituting your own information for the items in boldface type:

```
[MsgBlk]
form=test1.fmx
userid=scott/tiger@iasdb
archive_jini=f90all_jinit.jar,OfsMessage.jar
archive_ie=f90all.cab,OfsMessage.jar
archive=f90all.jar,OfsMessage.jar
formsMessageListener=oracle.forms.iserver.MessageListener
recordFileName=c:\temp\is
```

3. When you begin to record, append a config parameter on the initial browser page's URL, as shown in boldface type in the following sample URL:
   `http://ntsap45b:7779/forms90/l90servlet`**`?config=MsgBlk`**

Oracle Forms Server Recording Options

QALoad records through your default browser.

Connections

Forms version: Select your Oracle Forms Server version. This parameter must match the version of WebForms you are using or you will not be able to record. You can select:

- 10g Servlet - for Oracle Application Server 10g

- 9i Servlet - for Oracle 9i Application Server
- 6i Servlet - for version 6i Servlet Applications
- 6i Servlet (11i) - for version 6i Applications running in the 11i e-BusinessSuite
- 6i Socket - for version 6i Applications in Socket mode
- 6.0 Socket - for Forms 6.0

Enable server-side recording: Select to enable Oracle Forms 9i server-side recording.

Listener servlet URL: Type the URL of the listener servlet that is used by the application under test.

Use SSL: Select this option if the application uses Forms Version 9i and the application is SSL-enabled.

Certificate file: For SSL mode only. Type the name of, or browse for, the Oracle Jinitiator's certDB file that is located on the client machine. The certDB file is used by Oracle 9i to verify the SSL Certificate Authority on the client side prior to the Forms connection.

Socket URL: For socket connections with Oracle Forms versions 6i or 6.0, enter the URL to initiate your applet. If you are recording from 9i, this field is unavailable.

Port: If your Oracle Forms version is anything other than 9i, type the port number on your Oracle Forms Server application server for QALoad to listen on. This is the same port as the Forms listener (usually 9000).

To determine the server port:

1. Start the WebForms application in the browser.
2. Once the application has started, choose the menu option **View>Source**.
3. Look in the source code for a line that resembles this:
   `<PARAM NAME = "serverPort" VALUE ="9000" >`. This value is the server port.

Additional Jar Files

Override application defaults: If you need to use Java resources other than those that are your application's defaults, select the check box and then click Add to navigate to the appropriate JAR files and add them to the list.

Oracle Forms Server conversion options

General

Stop running when server messages indicate errors: Select to check server messages for errors. If this option is selected, the script stops running if errors are encountered. If this option is not selected, the script ignores errors and continues.

Stop only if server message matches specified string: Select to allow script failure only if the specified server message is received. This option is available only if the Stop running when server messages indicate errors is also selected.

Message: Type a string to match against server messages, and select one of the following match types:

contains: Match based on the specified characters appearing anywhere in the message.
is exactly: Match based on the specified complete error message name.
begins with: Match based on the specified characters appearing at the beginning of the message.
ends with: Match based on the specified characters appearing at the end of the message.

Socket Mode

Send heartbeat every [n] minutes: Type the number of minutes between each Forms 6i heartbeat message. Compuware recommends using a value of 4 or more to prevent socket usage issues.

Simulate an Oracle Applications 11i Login: Select to simulate an Oracle Applications 11i login via a Personal Home Page. Then complete the following three fields:

HomePage URL: Type the location (URL) of the Personal Home Page.
Userid: Type the user name to be used on the Personal Home Page.
Password: Type the password for the user name that is specified above in the Userid field.

Post Capture Options: Opens the OFS Post Capture Options dialog box where you can process a WWW file that is hexadecimal encoded (proxycap). This produces a new cap file, postcapweb file, longcap file, and sortedcap. This process overwrites these files when they already exist.

Note: Before using Post Capture processing, you must select the appropriate Forms version in the Oracle Forms Server Record Options dialog box.

Checkpoints in Oracle Forms Server scripts

EasyScript for Oracle Forms Server supports QALoad's automatic middleware checkpoint timings in both HTTP and socket modes. Default checkpoints (Begin/End Checkpoint pairs) are not supported.

Automatic checkpoints are enabled from the Conductor's Timing Options column on the Script Assignment tab and are enabled on a script-by-script basis.

At playback, automatic checkpoints are executed during the ofsSendRecv statement.

Forms validation/playback debugging options

Debug data

When the Debug Data option is enabled on the Configure Script Development Workbench dialog box for validation, or the Conductor's Debug Trace option is enabled for playback, executed script statements will be displayed. For example:

```
VU 0 : Line:90, ofsSetWindowSize( "FORMWINDOW" ,6, OFS_ENDMSG, 137, 750, 600 )
VU 0 : Line:91, ofsActivateWindow( "WINDOW_START_APP" ,11, OFS_ENDMSG, 247 )
VU 0 : Line:92, ofsShowWindow( "WINDOW_START_APP" ,11, OFS_ENDMSG, 173 )
VU 0 : Line:93, ofsFocus( "BUTTON" ,51, OFS_ENDMSG, 174 )
VU 0 : Line:94, ofsSetWindowSize( "FORMWINDOW" ,6, OFS_ENDMSG, 137, 750, 600 )
VU 0 : Line:95, ofsSendRecv( 1 )  //ClientSeqNo=2|MsgCount=6
```

Oracle Forms Server method reference

QALoad provides descriptions and examples of the various methods and functions available for an Oracle Forms Server script. For details, refer to the Language Reference Help section for Oracle Forms Server.

Using the certDB File for OFS Replay

In some Oracle Forms Server environments, the certificates needed for the SSL handshake with the server are not in the default wallet used during replay. This causes the SSL handshake to fail.

You can use the certDB file used by JInitiator for OFS replay. To do this for Script Validation, place the certdb.txt file in the BinaryFiles directory.

To use the certDB file for OFS replay in Conductor:

1. In the Conductor's Script Assignment tab, select a script in the Script column.
2. Click Browse . The External Data dialog box appears.
3. In Attached Files, click **Add**. The Add Attached File dialog box displays.
4. From the BinaryFiles folder, select **certdb.txt**, then click **Open**.
5. Click **OK**.

Advanced Scripting Techniques

Understanding the C++ script

Oracle Forms Server scripts are produced for Oracle Forms, 6.0, 6i, and 9i (Release 2 and later) recordings. The C++ script executes OFS-related statements by passing the statements in the script DLL to the OFS Java engine that performs the client activities and the client communication with the server. Because the C++ script statements are directly tied to corresponding methods in the OFS Java engine, modifications to the script statements are limited to changing the property parameter values through variablization.

An OFS C++ script contains three main sections: Connection, Application Body, and Disconnect. The QALoad transaction loop includes all three sections by default. The transaction loop can be moved using the guidelines described in Moving the OFS transaction loop. An internal auto checkpoint is created during connection statements and transmission statements.

The C++ script statements are a condensed version of the Java-style script statements. The C++ script statements show the GUI controls in the OFS application and the control properties, which are either control attributes or activities. For example:

```
ofsClickButton( "BUTTON", 52, OFS_ENDMSG, 325 );
```

In this example, the user clicks (property 325) a button (control ID 52). OFS_ENDMSG is a flag that indicates that the GUI activity ends the current OFS Message.

QALoad also allows OFS and WWW statements from a Universal session to be scripted in the C++ script, providing the ability to play back WWW and OFS statements.

Connection statements

The connection script lines in the C++ script vary depending on the type of Forms connection mode that is active. You choose the Forms connection mode on the Oracle Forms Server Recording Options dialog box. Forms connection modes include server-side recording, HTTP, HTTPS, or socket.

Server-side recording is limited to applications that use Forms 9i (applications running in Oracle 9iAS Release 2 and above). HTTP connection mode is available for applications using Forms 9i and for applications using the patched Forms 6i version configured with the HTTP servlet. HTTPS connection mode is strictly for SSL-enabled applications that use Forms 9i. Socket connection mode is for applications that use Forms 6i and lower versions, such as Oracle 11i.

Server-side recording connections

Server-side recording mode contains only one connection statement. The function that is used – ofsSetServletMode – contains the listener servlet value that you entered on the Oracle Forms Server Recording Options dialog box. The first parameter defines the HTTP or HTTPS configuration of the application environment. The second parameter defines the name of the Forms Listener Servlet used by the application. To connect, QALoad internally invokes Oracle's dispatch calls using the two parameters. Oracle's proprietary classes provide the implementation for the HTTP or HTTPS connection. For example:

```
ofsSetServletMode(OFS_HTTP, "http://ntsap45b:7779/forms90/l90servlet" );
```

HTTP connections

HTTP connection mode contains multiple connection statements. To connect, QALoad internally performs Java calls to accomplish the following tasks:

- Define HTTP header properties
- Connect to the Forms Servlet (an HTTP-GET request)
- Set the parameters of the Forms Listener Servlet
- Connect to the Forms Listener Servlet (an HTTP-GET request)
- Set additional HTTP header property for the Listener Servlet
- Connect to the Forms Listener Servlet (an HTTP-POST request). The last connection statement also initiates the required Forms “handshake” and determines the Forms encryption used by the application environment.

For example:

```
ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.9" );
ofsHTTPSetHdrProperty("Host", "ntsap45b:7779" );
ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, "*/*; q=.2"
);
ofsHTTPSetHdrProperty("Connection", "Keep-alive" );
ofsHTTPConnectToFormsServlet(
"http://ntsap45b:7779/forms90/f90servlet?ifcmd=startsession"  );
ofsHTTPSetListenerServletParms( "?ifcmd=getinfo&ifhost=C104444D01&ifip= "192.168.234.1"
);
ofsHTTPConnectToListenerServlet( "http://ntsap45b:7779/forms90/l90servlet");
ofsHTTPSetHdrProperty("Content-type", "application/x-www-form-urlencoded" );
ofsHTTPInitialFormsConnect();
```

HTTPS connections

HTTPS connection mode uses the same connection statements as HTTP mode. To connect, QALoad internally performs the same tasks as the HTTP connection mode plus it performs the SSL connection when the ofsHTTPSDoSSLHandshake function is called. This statement is positioned in the script before the ofsHTTPConnectToFormsServlet function.

Socket connections

Socket mode contains only one connection statement. The function that is used – ofsConnectToSocket – contains the port number and the URL you entered on the OFS Recording Options dialog box to start OFS capture. The port value is the port on which the Forms Server directly listens for Forms traffic. To connect, QALoad uses Java calls to open a Java socket using the parameters, initiate the required Forms "handshake”, and determine the Forms encryption used by the application environment. For example:

```
ofsConnectToSocket("10.10.0.167", 9002 );
```

Application statements

The application statements in the C++ script consist of property statements and transmission statements. Property statements describe the attributes and activities of GUI controls in the application. Transmission statements send the GUI controls and their properties as Forms Message data to the server. There is only one transmission statement: ofsSendRecv. QALoad creates an internal auto checkpoint when this statement is executed. In the following example, the first two (property) statements set the location and size of a FormWindow GUI control. The ofsSendRecv statement sends the GUI control properties to the server.

```
ofsSetWindowLocation( "FORMWINDOW", 6, OFS_ENDMSG, 135, 0, 0);   //Property
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 650, 500);   //Property
ofsSendRecv(1 ); //Transmission
```

Parameters of a property statement:

The parameters of a property statement are arranged in the following sequence:

1. **Captured control name**. If the name is not available, this value is the class name to which the control belongs.
2. **Captured control ID**.
3. **Action type**. This flag indicates if the property is to be added to the current Forms Message or if the property ends the current Forms Message. During playback, each control is treated as a Forms Message. When the current Message ends, *QA*Load translates the control and its properties to binary format. The valid values are:
   - OFS_ADD – add the property to the current Message.
   - OFS_ENDMSG – add the property to the current Message and end the Message.
   - OFS_STARTSUBMSG – add the property of the succeeding nested Message to the current Message.
4. **Property ID**. The Forms version-specific ID of the property.
5. **Property value**. Captured value of the property (optional)
6. **Property value**. Captured value of the property (optional)

For example:

```
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 650, 500);
```

In this example, control ID 6, which belongs to GUI class FORMWINDOW, is resized (PROPERTY 137) to have coordinates 650 and 500. This marks the end of the current Message.

Forms environment statements:

The initial set of statements in the Forms script describes the Forms application environment. In this set, the "version" and the "cmdline" properties are the most important. The version property shows the Forms Builder version used by the application. The version indicates the capabilities of the application. For example, some versions cannot support HTTP connections. The cmdline property shows the Forms configuration parameters passed to the server by the Forms applet. The parameter "record=names" indicates that the application enables GUI control names to be captured. Control names are preferred in multi-threaded playback. The "ICX" parameter indicates that the application uses a Personal Home Page, which requires that you supply OracleAppsLogin information on the Oracle Forms Server Convert options dialog box for the script to run successfully.

In the sample script below, the Forms builder version is 90290 (the version used in Oracle 9iAS Release 2, unpatched). The cmdline property shows "record=forms" which defaults "record=names". The cmdline property does not have the "ICX" ticket parameter.

```
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "90290" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1024, 768);
ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
 "server module=test1.fmx userid=  sso_userid=  debug=no buffer_records=no debug_"
 "messages=no array=no query_only=no quiet=yes render=no host=ntsap45b.prodti.com"
 "puware.com port= record=forms tracegroup=debug log=run1 term=" );
ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "0" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "8421504" );
ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 8, 20);
ofsSetNoRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
ofsSetPropertyString( "RUNFORM", 1, OFS_ENDMSG, 530, "America/New_York" );
ofsSendRecv(1 );
//ClientSeqNo=1|CapTime=1086884188.281|MsgCount=1
```

Sending messages to the server:

The ofsSendRecv statement sends the accumulated GUI controls and their properties to the Forms Server as binary data. This statement represents the point at which the client sends a Forms Terminal Message to the server. In Oracle Forms, the client and the server must end each data block with a Terminal Message before any transmission occurs.

Internally, QALoad varies the binary data transmission depending on the connection mode:

- For server-side recording mode, *QA*Load sends the binary data by invoking Oracle's dispatch calls. Oracle's own classes provide the implementation for the HTTP transmission.
- For HTTP or HTTPS mode, *QA*Load wraps the binary data inside an HTTP stream and invokes Java's HTTP calls.
- For socket mode, *QA*Load sends the binary data directly to the Java socket opened at the connection point.

The ofsSendRecv statement has one parameter: the response code of the captured Terminal Message. The possible values for this parameter are 1 (add), 2 (update), and 3 (close). Typically, when the response code is 3, the Forms Server reacts by removing the GUI controls associated with the client message from the server cache.

A comment line appears after each ofsSendRecv statement that contains script-tracking information. The information on the comment line is also found in the capture file in each ofsSendRecv capture line. The comment line shows the relative sequence of each client request, as represented by a Terminal Message, from the start of the application (e.g. ClientSeqNo=1). The comment line also shows the timing mark of the captured Terminal Message (e.g. CapTime=1086884188.281) and the number of Forms messages contained in the request (e.g. MsgCount=1). The number of Messages can be verified by counting the preceding ENDMSG and STARTSUBMSG flags in the request block. The comment line is useful for debugging playback issues because it readily shows the client request sequence number where the issue is occurring.

Getting the server reply:

During the execution of ofsSendRecv, QALoad also obtains the server's reply and translates the binary Forms data into Forms control values and control properties. The values are also written to the playback log file (in capture file format) if script logging is enabled. The following sample is a server reply:

```
VU 0 :    M|S|2|0|1
VU 0 :    P|S|322|java.lang.Integer|0|151000320
VU 0 :    P|S|279|java.lang.Boolean|0|false
VU 0 :    P|S|525|java.lang.String|AMERICAN_AMERICA.WE8MSWIN1252
VU 0 :    T|S|1|ServerSeqNo=1|MsgCount=76
```

The first line indicates the start of a Forms Message from the server (M|S). The third parameter is an action code (1= add, 2= update, 3= delete, 4= get property value). The fourth parameter is the Class Code of the control (0 = root class). The fifth parameter is the Control ID (1= RunForm).

The second, third and fourth lines are property lines related to the above Forms Message from the server (P|S). The third parameter of each line is the property ID (322). The fourth parameter is the data type of this property (java.lang.Integer). The fifth parameter is the data value. If the value is 0, the data value is in a sixth parameter (false).

The third line is the terminal message line from the server (T|S). The third parameter is the response code associated with the terminal message (1= add, 2= update, = close). The fourth parameter is the relative sequence of the server reply, as represented by a Terminal Message, from the start of the application (e.g. ServerSeqNo= 1). The fifth parameter is the number of Forms messages contained in the reply (e.g. MsgCount = 1). The number of Messages may be verified by counting the preceding M|S flags in the reply block. The fourth and fifth parameters are script-tracking information, which can be useful for debugging a playback issue. If logging is enabled, the log file shows the tracking information, which can make the comparison between server responses and captured responses easier.

Processing large data and delayed response scenarios:

When HTTP or HTTPS connection mode is used, Forms data is wrapped inside the HTTP reply stream. QALoad checks the HTTP header of the reply before processing the Forms data. The HTTP header sometimes indicates that the client needs to perform additional HTTP POST requests to obtain the complete Forms data. This indication occurs when the content-length of the reply is 64000 (a large data scenario), or the content-type is "text/plain" and the HTTP header contains an "iferror: " string (a delayed response/re-post scenario). QALoad performs the necessary POST requests to obtain the complete reply data, and then translates the accumulated reply data to Forms controls and properties.

Disconnect statements

The disconnect script lines vary depending on the Forms connection mode.

- In server-side recording mode, the ofsServerSideDisconnect script statement internally invokes Oracle's dispatch calls to disconnect.
- In HTTP mode, the ofsHTTPDisconnect statement internally makes Java calls to disconnect the main URL connection from the servlet.
- In socket mode, the ofsSocketDisconnect statement closes the socket on which the Forms Server listens for traffic.

Using script logging as a debugging tool

You can debug a playback issue in a C++ script by enabling replay logging. The option for enabling replay logging is located on the Script Assignment tab of the Conductor. For more information about enabling log file generation, see Debugging a script.

In Java-based scripts, logging is not enabled by default. To enable logging, change the parameter of the Logging method to true in the script. For example:

```
oracleForms.Logging( true );
```

When logging is enabled, QALoad writes the client requests and server replies to the playback log file in the same format as the capture file. The playback log file is found in the `\QALoad\LogFiles` directory. When there is an issue during playback, such as the server not responding to a client request, you can compare the capture files and check the differences in the server reply data. Both the capture file and the log file contain tracking information appended to the server's terminal messages. The tracking data contains the relative sequence number of the server reply from the start of the Forms session and the timing mark. The tracking data also shows the number of Forms messages contained in the reply block. The number of messages are based on the number of "M|S" lines prior to the "T|S" lines.

In the following example, the first set of statements shows the logged statements and the second set of statements shows the captured statements. The ServerSeqNo value shows that this is the 8th reply from the server. The MsgCount value of 1 shows that only one Forms Message is included in this reply block.

```
1087419810.000|ofsShowWindow|WINDOW_START_APP|11|OFS_ENDMSG|173|PROPERTY_VISIBLE|java.lang.B
oolean|true
1087419810.000|ofsSendRecv|1|ClientSeqNo=8|CapTime=1087419810.000|MsgCount=1
1087419810.000|M|S|2|0|30
1087419810.000|P|S|135|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087419810.000|P|S|137|java.awt.Point|0|java.awt.Point[x=706,y=464]
1087419810.000|P|S|139|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087419810.000T|S|1|ServerSeqNo=8|CapTime=1087419810.000|MsgCount=1
```

```
1087402349.296|ofsShowWindow|WINDOW_START_APP|11|OFS_ENDMSG|173|PROPERTY_VISIBLE|java.lang.B
oolean|true
1087402349.296|ofsSendRecv|1|ClientSeqNo=8|CapTime=1087402349.296|MsgCount=1
1087402349.296|M|S|2|0|30
1087402349.296|P|S|135|java.awt.Point|0|java.awt.Point[x=0,y=0]
1087402349.296|P|S|137|java.awt.Point|0|java.awt.Point[x=706,y=464]
1087402349.296|P|S|139|java.awt.Point|0|java.awt.Point[x=0,y=0]
```

```
1087402349.296|T|S|1|ServerSeqNo=8|CapTime=1087402349.296|MsgCount=1
```

Moving the OFS transaction loop

To enable movement of the QALoad transaction loop in the C++ script, you must first record a full business transaction and a partial business transaction. The business transaction is the activity that you would like to repeat during QALoad playback. Insert QALoad capture comments (using the Insert Command button on the Recording toolbar) at the start and end of a business transaction. These comments will help you find the spots in the script where you would like to reposition the BEGIN_TRANSACTION() and END_TRANSACTION() statements. Then re-start the business transaction.

QALoad's OFS script presents a sequence of Forms GUI objects. The GUI objects contain context dependencies. For example, when a window is opened, the buttons, text fields and edit boxes inside that window are logically dependent on the state of that window. When only one business transaction is captured and the corresponding script's transaction loop is moved, the sequence of the GUI objects is broken during the second iteration of the transaction loop. The broken sequence results in a broken context, which causes the server to respond unpredictably during playback on the second and subsequent iterations of the transaction loop. When the business transaction is restarted during capture, the Forms GUI objects that compose the new transaction are used to anchor into the new transaction loop without breaking the context dependencies of GUI objects.

When modifying the script, use the comment lines as guides in moving the END_TRANSACTION() and BEGIN_TRANSACTION() statements. Ensure that there is a contextual flow from the new position of the END_TRANSACTION() statement to the new position of the BEGIN_TRANSACTION() statement. The set of GUI objects that belong to the ofsSendRecv() statement just before the new END_TRANSACTION() statement must be the same as the set of GUI objects that belong to the ofsSendRecv() statement prior to the new BEGIN_TRANSACTION() statement.

During playback, modify the Conductor setting for Transaction Pacing on the Script Assignment tab to allow the database to process each new business transaction.

The following example shows a modified OFS transaction loop:

New position of the BEGIN_TRANSACTION statement

```
/*
NewSales
*/

DO_SLEEP(13);
ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "B" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 1, 1);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "1" );
ofsIndexKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 175, 97, 0);

DO_SLEEP(6);
ofsSendRecv(1); //ClientSeqNo=31|MsgCount=2|1093981339.921
BEGIN_TRANSACTION();

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "Business World" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 14, 14);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsRemoveFocus( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 174 );
ofsSetSelection( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ADD, 195, 0, 0);
ofsSetCursorPosition( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ENDMSG, 193, "0" );
ofsFocus( "ORDER_CUSTOMER_NUMBER_0", 564, OFS_ENDMSG, 174 );

DO_SLEEP(6);
ofsSendRecv(1); //ClientSeqNo=32|MsgCount=4|1093981347.296
```

New position of the END_TRANSACTION statement

```
/*
EndTrans
*/

DO_SLEEP(39);
ofsSendRecv(1); //ClientSeqNo=61|MsgCount=4|1093981458.031

ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsSelectMenuItem( "Sales Orders", 257, OFS_ENDMSG, 477, "MENU_11059" );

DO_SLEEP(26);
ofsSendRecv(1); //ClientSeqNo=62|MsgCount=2|1093981485.265

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "B" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 1, 1);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "1" );
ofsIndexKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 175, 97, 0);

DO_SLEEP(3);
ofsSendRecv(1); //ClientSeqNo=63|MsgCount=2|1093981488.437
END_TRANSACTION();

ofsEdit( "ORDER_SOLD_TO_0", 562, OFS_ADD, 131, "Business World" );
ofsSetSelection( "ORDER_SOLD_TO_0", 562, OFS_ADD, 195, 14, 14);
ofsSetCursorPosition( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 193, "14" );
ofsIndexSKey( "ORDER_SOLD_TO_0", 562, OFS_ENDMSG, 176, 10, 0);

DO_SLEEP(13);
ofsSendRecv(1); //ClientSeqNo=64|MsgCount=2|1093981502.640
```

Tips:

During capture, the OFS configuration parameter "record=names" must be enabled to produce control names that may be included in the converted script. Control names persist throughout the Forms session, unlike control IDs, whose values may change at runtime. Add the “record=names” parameter in the `Formsweb.cfg` file or add this parameter to the startup servlet URL. Control IDs can create problems when the transaction loop is moved. Some of the control IDs that have been instantiated by the server prior to the new transaction loop lose context during iterations of the new loop. For example, in a second loop iteration, the server assumes that these client controls are new, generates new control IDs, and eventually cannot find the proper context. Then the server stops responding. If control names are used, Forms objects that have been instantiated before the new transaction loop are maintained through all iterations of the loop because the control name persists throughout the application session.

During playback, ensure that the sleep factor is at 100% and that the transaction pacing is set to a large enough value for the server to process the business transaction that is contained in the new loop. These options can be set on the Script Assignment tab of the Conductor.

OFS and WWW Universal sessions

You can record with a Universal session to capture both the OFS and WWW transactions and merge the two sets of transactions into one script. The captured WWW statements contain non-servlet, non-Forms data such as GIF objects, while the captured OFS statements contain the Forms data.

Universal scripting for OFS-WWW sessions is available in C++ format only. After conversion, the WWW statements do not appear in visual scripts.

Note: The only Universal session combination that is available for Oracle Forms Server is the combination of WWW and Oracle Forms Server.

When an Oracle Applications login is captured, the login can be scripted using the OracleAppsLogin statement or the ofsSetICXTicket statement. Compuware recommends that you use ofsSetICXTicket.

When OracleAppsLogin is used, the login is performed twice: once by the scripted DO_Http statement for the WWW actions and again by OracleAppsLogin. To prevent duplicate logins, you must comment out the DO_Http (WWW middleware) statement.

When ofsSetICXTicket is used, the login is performed just once. This statement allows the WWW login to execute, extracts the ICX ticket from the server reply, and passes the ICX ticket to the Forms session.

To use ofsSetICXTicket, you must modify the script.

To capture an Oracle Applications login with ofsSetICXTicket:

1. Add the following variable declaration statements to the top of the script:
```
char *p;
char ICX_Ticket[100];
char *pTicket;
```
2. In the `*.postcapweb` file, find the HTTP request that returns the ICX ticket. The reply should contain a string that indicates the ICX ticket value, such as `"ICX_TICKET="`. Note the left and right characters that delimit the ICX ticket value. In the example in step 4, the left delimiter is `"icx_ticket='"` and the right delimiter is `"'"`.
3. In the script, find the matching request line for the HTTP request.
4. After the matching HTTP request line, add the DO_GetUniqueString statement using your chosen delimiters. For example:
```
p = DO_GetUniqueString( "icx_ticket='", "'");
```
5. Add script lines that copy the extracted value into your script variables.
```
strcpy(ICX_Ticket, p);
pTicket=ICX_Ticket;
```
6. (optional) Verify the ICX ticket value.
```
RR__printf("ICX_Ticket=\"%s\"\n", ICX_Ticket);
```
7. Add the script line `ofsSetICXTicket(&pTicket);` before the ofsInitSessionCmdLine. This passes the value of the ICX ticket to the ofsInitSessionCmdLine statement.
8. Free the memory allocated by DO_GetUniqueString and ofxSetICXTicket before the end of the transaction:

```
free(p);
p=NULL;
free(pTicket);

pTicket=NULL
```

SAP

Linking errors during validation or compilation of SAP scripts

When you re-record SAP 4.x scripts for SAP 6.20/SAP 6.40, you must click the Build SAP Libraries button on the SAP Convert Options dialog box. This button generates new libraries based on the version of SAP that is currently installed. If you have upgraded to a newer version of SAP and do not update the libraries, you may experience various linking errors during validation or compilation.

SAP script validation fails

If your SAP script fails during validation, you may need to disable automatic proxy configuration in Internet Explorer.

To disable automatic proxy configuration:

1. In Internet Explorer, click **Tools>Internet Options**.
2. On the **Connections** tab, click the **LAN Settings** button.
3. Ensure that the **Use automatic configuration script** check box is cleared.

If disabling the automatic proxy configuration does not solve the problem, consider increasing the script execution timeout value to 100 seconds or to the length of the capture file (in seconds), whichever is greater.

To increase the timeout value:

1. With an SAP session open in the Script Development Workbench, click **Options>Workbench**.
2. On the **Script Validation** tab, type the new value in the **Wait up to** field.
3. Click **OK**.

SAP 4.x

Overview

Use QALoad's SAP middleware support to load test systems that run SAP 4.0B, 4.5 or 4.6D.

What is SAP?

The SAP GUI front end is a middleware that allows user to access SAP servers from Windows. The SAP servers run various SAP business applications, such as applications for customer relationship management, human resources, and supply chain management.

Connecting to the SAP server

First, you must connect to the SAP server. Once you have connected to a machine that is running the SAP server, you can log on and interact with the SAP applications.

Recording scripts

To record scripts with SAP 4.x, you must start the QALSAP application. The QALSAP application enables you to connect and log on to the SAP server.

SAP recording options (Versions 4.x)

Before you can successfully record transactions from a SAP-based application, you must select the Dialog (modal) option on the Help>Settings>F4 Help tab in the SAP application. You must do this for the user you are recording.

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording

Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead.

Automatic: Select this option for QALoad to automatically start your application for recording, allowing you to record early application startup activity. This is the recommended method of recording, as it takes advantage of QALoad's enhanced abilities to handle various multithreaded programming techniques.

Command Line: Enter or browse for the path to QALSAP, then enter the startup parameter \c. For example, enter: `c:\Program Files\Compuware\QALoad\Qalsap.EXE \c`.

The following additional startup parameters are available:

Parameter	Description
`\a`	Version 3.1 clients only. Use this parameter in place of `\c` if QALSAP cannot connect to your Sapgui. QALoad will then record directly from QALSAP. For example: `c:\Program Files\Compuware\QALoad\Qalsap.EXE \a`
`\m`	Minimizes the Sapgui window after successfully logging in. For example: `c:\Program Files\Compuware\QALoad\Qalsap.EXE \c \m`
`\r#`	Where # is a number from one to five. Type `\r` followed by a number from one to five to indicate how many times QALoad should attempt to login to Sapgui if the first attempt times out. For example: `c:\Program Files\Compuware\ QALoad \Qalsap.EXE \c \r5`

Working Directory: Enter the working directory of your SAP application.

Note: If you entered the full path in the Command Line field, this field is filled in automatically.

Disable SAP Enjoy During Capture: Select this check box to disable SAP Enjoy (SAP 4.6 and above) before recording. If you choose not to disable SAP Enjoy, you must manually shut down the SAP client tray application before you stop recording.

Starting QALSAP

QALSAP is the application that enables recording from QALoad for SAP 4.x scripts. To begin recording, you must start this application.

To start QALSAP:

1. From the Windows **Start** menu, choose **Run**.
2. On the **Run** dialog box, type `QALSAP`.

SAP conversion options (Versions 4.x)

Consolidate Checkpoints: Select to consolidate checkpoints with the same description into a single checkpoint. The checkpoint description is the Tcode concatenated with the first 30 characters of the screen title.

If you do not select this option, and QALoad detects a duplicate description, it will append a sequence number after the checkpoint description.

Graphical User Interface: Select to view validation in graphical mode. If selected, the script will step through validation transactions with SAPGUI running. Due to memory requirements (15-20 MB per user), select this option only if you are replaying a single user.

Animating an SAP capture file

Animating a capture file plays back the series of transactions from an SAP recording.

To animate an SAP capture file:

1. Open an SAP session in the *QA*Load Script Development Workbench.
2. In the Workspace Pane, click the Captures tab.
3. Select the capture file you want to animate.
4. From the **Session** menu, choose **Animate>Start**. The *QA*Load Script Development Workbench opens the file in QALSAP, where you can view each transaction graphically.

Viewing SAP 4.x post-test log files

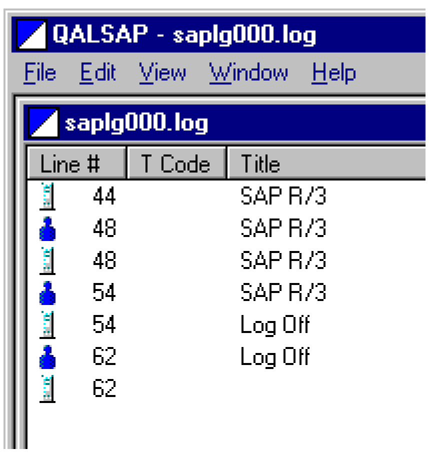
If you selected the Detailed Logging option on the QALoad Script Development Workbench's SAP Conversion Options dialog box before you ran the test, QALoad automatically generated a log file for each virtual user named `saplg###.log` (where ### is the virtual user number) during the test. Each log file contains a graphical representation of the events sent to and received from the server for a particular virtual user.

You can open and delete a log file from the QALoad Script Development Workbench using the following procedure.

To open a virtual user log file:

1. With an SAP session open in the *QA*Load Script Development Workbench, select **File>Browse**.
2. On the **Browse** dialog box, double-click **Log Files**.
3. After QALSAP opens, select **File>Open**.
4. In the **Files of Type** field, select **Log File**. The **Browse Log Files** dialog box opens, displaying the available SAP log files.
5. Double-click on the log file you wish to open. Log files are named `saplg###.log`, where ### is the virtual user number.

   QALSAP opens the selected log file. In the Line # column of the log file, each request by the client is marked by a blue "client" icon, while each response from the server is marked by a white "server" icon, as shown in the following image.



Viewing request or response details

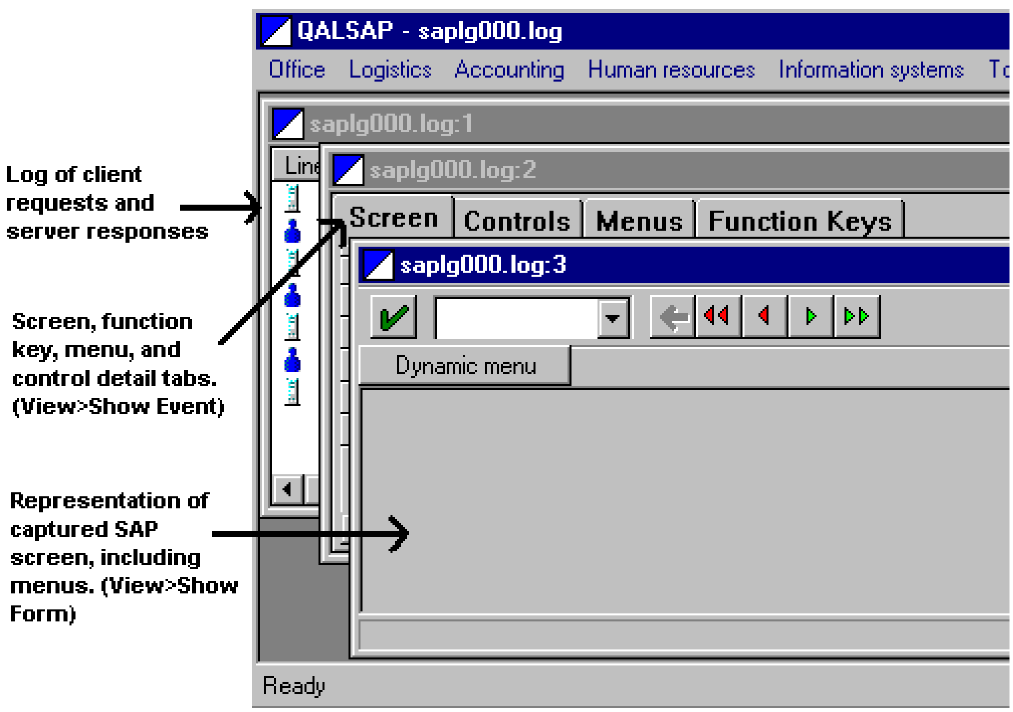
QALSAP allows you to view detailed information about each request and response in the log file, including each logged SAP screen, and its menus, function keys, and controls.

You determine which information to display by selecting one or both of the following commands from the View menu:

- **View>Show Event** — Opens a detail window, displaying detailed information about each logged event, including screen names, key names, tool tip text, and so on.
- **View>Show Form** — Opens a form window, displaying a graphic representation of the logged SAP screen and related menus.

To view a specific request or response:

1. With a log file open in QALSAP, select the appropriate command(s) from the **View** menu to determine what amount of detail to view.
2. Double-click on the line number of the request or response you want to view. Detail windows open for the selected request or response, depending on the options you set in the **View** menu. In the image below, both **Show Event** and **Show Form** were selected.
   - To view the next like response—for example, if you are viewing a client request and want to view the next client request—select View>Goto Next Response. Alternately, from the form window, click the appropriate toolbar button to view the next or previous response.
   - To view the next event in the log, select View>Goto Next Event. Alternately, from the form window, click the appropriate toolbar button to view the next or previous event.



SAP 4.x command reference

QALoad provides descriptions and examples of the various commands available for an SAP script. For details, refer to the Language Reference Help section for SAP 4.x .

SAP 6.x

Overview

Use QALoad's SAP middleware to load test systems that run SAP 6.20 and 6.40.

What is SAP?

The SAP GUI front end is a middleware that allows users to access SAP servers from Windows. The SAP servers run various SAP business applications, such as applications for customer relationship management, human resources, and supply chain management.

Connecting to the SAP server

Once you have connected to a machine that is running the SAP server, you can log on and interact with the SAP applications.

Configuring an SAP client for load testing

Before you can record an SAP session, you must have an SAP client that is configured to enable QALoad to access the SAP server. Configure the SAP client through the SAP Logon application.

To configure an SAP client for load testing:

1. Start the SAP Logon application. From the taskbar, click **Start>Programs>SAP Front End>SAPlogon**.
2. Click the **New...** button on the SAP Logon dialog box. The New Entry dialog box appears.

3. Type values in the **Description**, **Application Server**, and **System number** fields.

   Note: QALoad uses the value in the Description field to connect to the server.

4. Click **OK**. The new SAP server entry appears in the list in the SAP Logon dialog box.

SAP recording options (Versions 6.x)

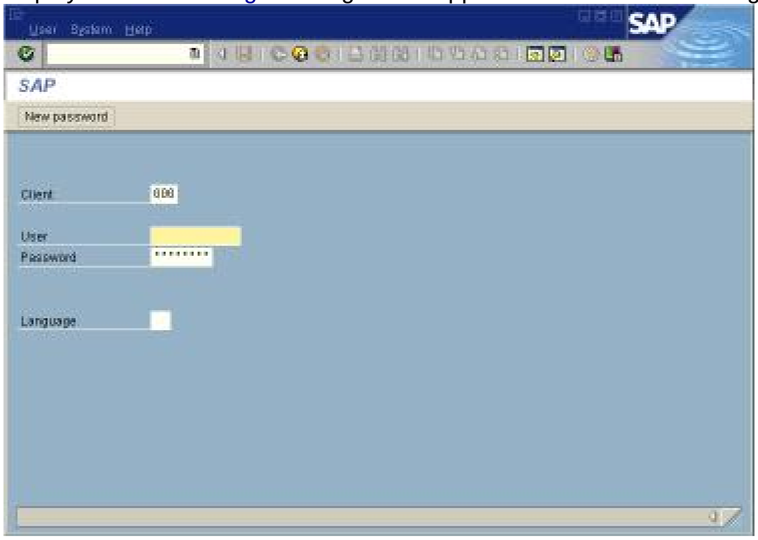
Save Server Description: Select to specify and save the server description (name) to which you want to connect during recording. If this check box is not selected, you are prompted for a server description during the log on process.

Recording an SAP session

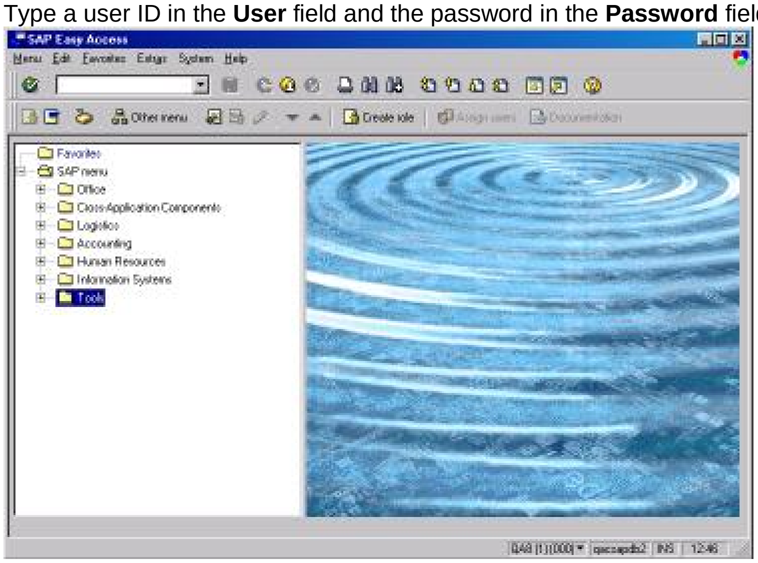
An SAP server connection must be configured before you can connect with QALoad. See Configuring an SAP client for load testing for more information. Additionally, your SAP administrator must set the `SAPGUI/User_Scripting` security profile parameter to TRUE to successfully record a script. For more information about SAP security settings, refer to the SAP publication titled "Sapgui Scripting Security".

To record an SAP session:

1. In the Script Development Workbench, click the **Record** button on the Session toolbar. If you have not already chosen SAP as the session type, click the **SAP Session** button to activate a new SAP session.
2. If you have not selected the Save Server Description record option, the SAP Server Description dialog box appears. Type the name of the SAP server to which you want to connect. This value is the same as the **Description** field that displays in the SAP Logon configuration application. Press **Enter**. A log on dialog box appears.



3. Type a user ID in the **User** field and the password in the **Password** field. Press Enter. The SAP application starts.



4. In the SAP application, turn off the scripting and notification options. Click the **Customizing of local layout button** and choose **Options**. The Options dialog box appears. On the Scripting tab, select **Enable Scripting**, but clear the two Notify check boxes.
5. Begin recording actions in SAP.

SAP conversion options (Versions 6.x)

Save Password: Select to save the encrypted password. If this check box is not selected, you are prompted for a password during conversion.

VB Script: Select to generate Visual Basic Script for debugging outside QALoad . If this option is not selected, you receive C++ scripts that can be used for playback within QALoad .

Build SAP Libraries: Click the Build button to generate the QALoad SAP libraries based on your version of SAP. If you receive linking errors while validating or compiling, you should click this button.

SAP 6.x command reference

QALoad provides descriptions and examples of the various commands available for an SAP script. For details, refer to the Language Reference Help section for SAP 6.x.

Advanced Scripting Techniques for SAP

Adding Custom Counters to Retrieve Server Information

The following example adds custom counters to obtain and save the SAP Server information that is available through the SAP Gui Scripting API. Notice that SAPGuiSessionInfo is called before logging off , because the data is not available after logging off.

```
int id1, id2, id3, id4;

long lRoundTrips,lFlushes;

// "Counter Group", "Counter Name", "Counter Units
// (Optional)", Data Type, Counter Type.

id1 = DEFINE_COUNTER("Cumulative Group", "Cumulative RoundTrips", 0, DATA_LONG,
COUNTER_CUMULATIVE);

id2 = DEFINE_COUNTER("Cumulative Group", "Cumulative Flushes", 0, DATA_LONG,
COUNTER_CUMULATIVE);

id3 = DEFINE_COUNTER("Instance Group", "Instance RoundTrips", 0, DATA_LONG,
COUNTER_INSTANCE);

id4 = DEFINE_COUNTER("Instance Group", "Instance Flushes", 0, DATA_LONG, COUNTER_INSTANCE);

SYNCHRONIZE();

BEGIN_TRANSACTION();

try{
   SAPGuiConnect( s_info,"qacsapdb2");
   ...
   SAPGuiSessionInfo(GetRoundTrips,lRoundTrips);
   SAPGuiSessionInfo(GetFlushes,lFlushes);
   SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
   SAPGuiCmd0(GuiButton,Press);
   SAPGuiCheckScreen( "SESSION_MANAGER", "SAPLSPO1", "Log Off" );

   COUNTER_VALUE( id1,lRoundTrips);
   COUNTER_VALUE( id2,lFlushes);
   COUNTER_VALUE( id3,lRoundTrips);
   COUNTER_VALUE( id4,lFlushes);

} // end try

catch (_com_error e){
   char buffer[1024];
   sprintf(buffer,"SAP: EXCEPTION 0x%x %s for VU(%i)\n",e.Error(), (char *)e.Description(),
S_task_id);

   RR__FailedMsg(s_info,buffer);

} // end catch

END_TRANSACTION();
```

Required commands

Certain commands must be present in an SAP script for it to run successfully. These commands are created automatically during the conversion process. Most of the commands exist before the BEGIN_TRANSACTION statement. The required commands include:

```
SET_ABORT_FUNCTION(abort function);

DEFINE_TRANS_TYPE("capture.cpp");

HRESULT hr = CoInitialize(0);

if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info, "ERROR initializing COM");

SAPGuiSetCheckScreenWildcard(‘*’);

SYNCHRONIZE();
```

Required commands for transaction restarting

When transaction restarting is enabled in the Conductor for an SAP script, the following commands, which are automatically added by QALoad during script conversion, must exist for the script to run:

```
SAPGuiApplication(RegisterROT);
SAPGuiApplication(RevokeROT);
SAPGui_error_handler(s_info, buffer);
```

The SAPGuiApplication command properly registers and removes the script's SAP GUI usage on the Runtime Object Table (ROT). If a transaction fails, these actions are taken to start and clean up the SAP environment.

> Note: Do not call RR__FailedMsg in an SAP script if the script includes a restart transaction operation. SAPGui_error_handler can be called with the same parameters as RR__FailedMsg to output a fatal error message while still allowing a proper clean up of the current transaction before restarting the transaction.

Error handling and reporting

A try/catch block is automatically generated for the commands between the BEGIN_TRANSACTION and END_TRANSACTION statements. This construct provides error handling and reporting from the script.

```
BEGIN_TRANSACTION();

try{

   SAPGuiConnect( s_info,"qacsapdb2");
   SAPGuiVerCheckStr("6204.119.32");

   //Set SapApplication = CreateObject("Sapgui.ScripingCtrl.1")
   //SapApplication.OpenConnection ("qacsapdb")
   //Set Session = SapApplication.Children(0).Children(0)

   DO_SLEEP(3);

   SAPGuiPropIdStr("wnd[0]");
   SAPGuiCmd3(GuiMainWindow, ResizeWorkingPane, 83, 24, false);

   DO_SLEEP(6);

   SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
   SAPGuiCmd1(GuiTextField,PutText,"qaload1");

   SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
   SAPGuiCmd1Pwd(GuiPasswordField, PutText,"~encr~1211616261");

   SAPGuiCmd0(GuiPasswordField,SetFocus);
   SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

   SAPGuiPropIdStr("wnd[0]");
   SAPGuiCmd1(GuiMainWindow,SendVKey,0);
   SAPGuiCheckScreen("S000","SAPMSYST","SAP");
```

```
    ...
    DO_SLEEP(10);
    SAPGuiPropIdStr("wnd[0]/usr/cntlIMAGE_CONTAINER/shellcont/shell/shellcont[0]/shell");
    SAPGuiCmd1(GuiCtrlTree, ExpandNode, "0000000003");
    SAPGuiCmd1(GuiCtrlTree, PutSelectedNode, "0000000004");
    SAPGuiCmd1(GuiCtrlTree, PutTopNode, "Favo");
    SAPGuiCmd1(GuiCtrlTree, DoubleClickNode, "0000000004");
    SAPGuiCheckScreen("SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access");
    SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
    SAPGuiCmd0(GuiButton,Press);
    SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");
} // end try
catch (_com_error e){
    char buffer[1024];
    sprintf (buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
             (char *)e.Description(), S_task_id);
    RR__FailedMsg(s_info,buffer);
} // end catch
END_TRANSACTION();
```

To include the log on within the transaction loop, move the SAPGuiConnect call inside the try block as shown in the following example:

```
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("capture.cpp");
RESULT hr = CoInitialize(0);
if( hr != ERROR_SUCCESS )
     RR__FailedMsg(s_info,"ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
BEGIN_TRANSACTION();
try{
    SAPGuiConnect( s_info,"qacsapdb2");
    SAPGuiVerCheckStr("6204.119.32");
    ...
    SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
    SAPGuiCmd0(GuiButton,Press);
    SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");
} // end try
catch (_com_error e){
    char buffer[1024];
    sprintf(buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
            (char *)e.Description(), S_task_id);
    RR__FailedMsg(s_info,buffer);
} // end catch
END_TRANSACTION();
```

To include the log on outside the transaction loop, move the log off section so that it follows the END_TRANSACTION statement. However, ensure that the recording within the transaction loop begins and ends in the same location in the menu system. For example:

```
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("capture.cpp");
HRESULT hr = CoInitialize(0);
```

```
if( hr != ERROR_SUCCESS )
   RR__FailedMsg(s_info,"ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');

SYNCHRONIZE();

SAPGuiConnect( s_info,"qacsapdb2");

SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
SAPGuiCmd1(GuiTextField,PutText,"qaload1");

SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~1211616261");
SAPGuiCmd0(GuiPasswordField,SetFocus);
SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow,SendVKey,0);
SAPGuiCheckScreen("S000","SAPMSYST","SAP");

BEGIN_TRANSACTION();

try{
   SAPGuiVerCheckStr("6204.119.32");
   ...
} // end try

catch (_com_error e){
   char buffer[1024];
   sprintf(buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
           (char *)e.Description(), S_task_id);
   RR__FailedMsg(s_info,buffer);

} // end catch

END_TRANSACTION();

SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
SAPGuiCmd0(GuiButton,Press);
SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");
```

Handling multiple logons

You may need to modify your script to handle multiple logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this issue, you can use the SAPGuiPropIdStrExists and SAPGuiPropIdStrExistsEnd commands to handle either scenario. This technique works by checking for the multiple logon dialog box from SAP and selecting the Continue option.

The following example demonstrates the usage of the SAPGuiPropIdStrExists and SAPGuiPropIdStrExistsEnd commands to handle multiple logons:

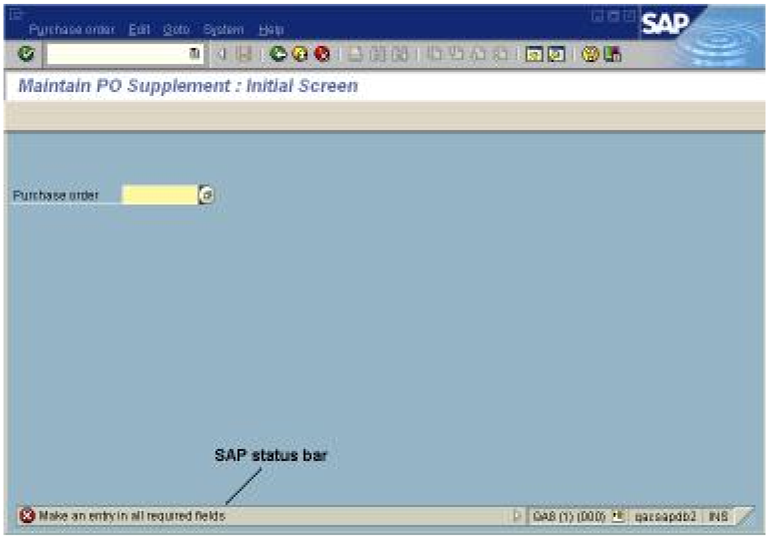
```
...

SAPGuiCheckScreen("S000","SAPMSYST","SAP");
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");

   DO_SLEEP(24);

   SAPGuiCmd0(GuiRadioButton,Select);
   SAPGuiCmd0(GuiRadioButton,SetFocus);
   SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");
   SAPGuiCmd0(GuiButton,Press);
   SAPGuiCheckScreen("S000","SAPMSYST","License Information for Multiple Logon");
```

SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");

...

Checking the SAP status bar

The SAP status bar displays error and status messages, as shown in the following figure.



You can use the SAPGuiCheckStatusbar command to test for certain status responses in the SAP environment.

The SAPGuiCheckStatusbar command is used in the following script example:

```
...
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow, SendVKey, 0);
SAPGuiCheckScreen("S000", "SAPMSYST", "SAP");
SAPGuiCmd3(GuiMainWindow, ResizeWorkingPane, 94, 24, false);

//SAPGuiCheckStatusbar returns TRUE if the message is found
//and FALSE if not found

BOOL bRetSts = SAPGuiCheckStatusbar("wnd[0]/sbar", "E: Make an entry in all required fields");

if (bRetSts)
    RR__printf(" True\n");

else
    RR__printf(" False\n");

...
```

Object life span

Whenever a script is run, all objects on the SAP GUI window are deleted and re-created. These objects, which are created in the SAP environment and can disappear without user interaction, can cause script failure if the script references the objects after they have disappeared.

For more troubleshooting information, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

UNIFACE

Uniface recording options

Uniface executable: Enter the full path or browse to the Uniface 7 executable that is used by the application you want to record. For example: `c:\usys72\bin\UNIFACE.exe.`

Working directory: Enter or browse to the working directory of your Uniface application or the directory of any additional files your application may require that do not reside in the application's path environment.

Initialization (.ini) file: Enter or browse to the full path to the Uniface application's initialization file.

Assignment (.asn) file: Enter or browse to the full path to the application's assignment file. For example: `c:\usys72\project\myapp.asn`.

Command line statement: Type command line options that should be used at application startup, including the command that is used to start the application. For example: `warehouse 1 control use=control dnp=tcp:`

Uniface conversion options

Includes: Type the full path or browse to the directory that contains the database include files.

Libraries: Type the full path or browse to the directory that contains the database library files.

Generate Uniface lists: Uniface can handle internal list structures. Select this check box to convert strings containing list items to a succession of DO_URB_xxx calls that manipulate Uniface lists.

Show output parameters: Select this option for the converted script to contain the output parameters of an operation as commented lines.

Insert trace messages as comments: Select this option for the converted script to contain the recorded content of the message frame as commented lines.

Uniface command reference

QALoad provides descriptions and examples of the various commands available for a Uniface script. For details, refer to the Language Reference Help section for Uniface.

Winsock

How QALoad handles DO_WSK_Send commands

QALoad displays the contents of a DO_WSK_Send command as a string in a Winsock script. Some of these strings are very large, which can cause a compiler error (fatal error C1076: compiler limit: internal heap limit reached) if there are several large strings in a single script.

To avoid this compilation error, QALoad does not allow strings that are displayed in a Winsock script to be more than 12,000 characters. If a DO_WSK_Send command has a send buffer larger than 12,000 characters, its buffer is broken into smaller strings during the conversion. These smaller strings are then copied into a char buffer named "SendBuffer", which is sent in the DO_WSK_Send command. The size of the SendBuffer variable, by default, is declared as the size of the largest DO_WSK_Send + 1000. For example:

```
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
```

```
/* Declare Variables */
char SendBuffer[22139];  //Largest send is 21139 + 1000

    ...
...
...
     strcpy(SendBuffer, "$$ ...."); //Assume a large string, shortened for this example
strcat(SendBuffer, "$$ ....");

     /* 12675 bytes */
DO_WSK_Send(S1, SendBuffer);

     ...
...
     strcpy(SendBuffer, "$$ ...."); //SendBuffer is reused
strcat(SendBuffer, "$$ ....");
strcat(SendBuffer, "$$ ....");

     /* 21139 bytes */
DO_WSK_Send(S1, SendBuffer);

     ...
...
     REPORT(SUCCESS);
EXIT();
return(0);
}
```

Handling Winsock application data flow

Frequently, server programs return unique values (for example, a session ID) that vary with each execution of the script and may be vital to the success of subsequent transactions. To create scripts that include these values, you need to substitute the hard-coded values returned by the server with variables. The following original and modified code examples demonstrate this technique.

Original code

In this script, the server sends a session ID in response to a connection by the client. This session ID is required to successfully complete subsequent transactions.

```
/*
* wsk-AdvancedTechniques_original.c
*
* This script contains support for the following
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);
```

```
#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;
{

/* Declare Variables */

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

/////////////////////////////////////////////////////
// The session id returned by the server is
// unique to each connection
/////////////////////////////////////////////////////

/* 21bytes: SessionID=jrt90847\r\n */

DO_WSK_Expect(S1, "\n");

//////////////////////////////////////////////////////
// This unique id is then used for subsequent
// requests
//////////////////////////////////////////////////////

/* 34 bytes */

DO_WSK_Send(S1, "SessionID=jrt90847\r\n:^B^@^@^@^B^@^@^@^A^@^@^@");

/* 15 bytes: ID Accepted#^@\r\n */

DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);
}

void abort_function(PLAYER_INFO *s_info)
{

RR__printf("Virtual User %i:ABORTED.", S_task_id);

EXIT();
}
```

Modified code

If the original script (wsk-AdvancedTechniques_original.c shown above) is replayed, it will fail because the session ID will not be unique; rather, it will be the session ID that is coded in the script. To use the unique session ID received from the server, variable substitution must be used.

```
/*
* wsk-AdvancedTechniques_modified.c
*
* This script contains support for the following
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;

{

/* Declare Variables */

char Buffer[64];
char SendBuffer[64];
int nBytesReceived = 0;

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

//////////////////////////////////////////////////////
// The reply from the server is read into
// the Buffer variable. We will then have
// the unique Session ID for this connection.
// Also need to null-terminate the buffer
// after receiving.
//////////////////////////////////////////////////////

DO_WSK_Recv(S1, Buffer, 64, 0, &nBytesReceived);
Buffer[nBytesRecieved] = '\0';

/* 21bytes: SessionID=jrt90847\r\n */

//DO_WSK_Expect(S1, "\n");
```

```
////////////////////////////////////////////////////////
// Finally, substitute the Session ID received from
// the server with the one coded in the script.
////////////////////////////////////////////////////////
sprintf(SendBuffer, "%s:^B^@^@^@^B^@^@^@^A^@^@^@", Buffer);
DO_WSK_Send(S1, SendBuffer);
/* 34 bytes */
//DO_WSK_Send(S1, "SessionID=jrt90847:^B^@^@^@^B^@^@^@^A^@^@^@");
/* 15 bytes: ID Accepted#^@\r\n */
DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);
END_TRANSACTION();
REPORT(SUCCESS);
EXIT();
return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
RR__printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
```

Winsock recording options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

> Notes:
> If you run a character-based application in a DOS window, the Script Development Workbench does not record the API calls.
>
> If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the Capture Initialization Phase check box on the Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box.

Automatic: Select this option if you want QALoad to automatically start your Winsock-based client, allowing you to record early application startup activity. This is the recommended method of recording because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. When you select this option, the QALoad Script Development Workbench records the API calls that occur before the client enters its message loop. Select this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

Command Line: Enter the command line of your Winsock-based client. Note that if you enter the full path, QALoad automatically enters the path in the Working Directory field.

Working Directory: Enter the working directory of your Winsock-based client, if necessary.

Capture: Select the Winsock version to record.

Set IP Addresses: Click this button to open the Add/Delete IP Addresses dialog box, which you can use to specify the IP addresses and ports on which you want to record Winsock API calls or that you wish to exclude from recording.

Winsock conversion options

There are no specialized conversion options for Winsock.

Winsock command reference

QALoad provides descriptions and examples of the various commands available for a Winsock script. For details, refer to the Language Reference Help section for Winsock.

Advanced Scripting Techniques for Winsock

Understanding data representation in the script

This section describes how data that is sent and received is displayed in a Winsock script. Use this section as a reference when you examine a script.

During the conversion process, QALoad determines how to represent each character in the script. This conversion process uses the following rules:

1. The character is compared to the “space” character in the ASCII table, which has a decimal value of 32. If the character’s value is less than 32, the following steps are taken:

   b. If the character is “\r”, “\n”, “\t”, or “\f”, it is represented in the script as a normal C escape character.

   c. If the character is either “^\” or “^^”, it is represented in the script as an octal character. For example, the values would be “\034” and “\036”, respectively.

   d. If the character’s value is less than 32 and it does not meet the descriptions in a) and b) above, it is represented in the script as a control character. For example, if the character is a null character, it is represented in the script as “^@”.

2. If the character’s decimal value is between 32 (the “space” character) and 126 (~), it displays in the script as a standard readable ASCII character, with the following exceptions:

   - If the character is “\ ”, which has a decimal value of 92, it is represented as “\ \” in the script.
   - If the character is “““, which has a decimal value of 34, it is represented as “\ ”” in the script.
   - If the character is “^”, which has a decimal value of 94, it is represented as “^^” in the script.

3. If the character has a decimal value of 127, which corresponds to Delete (DEL), it is represented as “^” in the script.

The following table summarizes the results of rules 1-3.

Code	Octal	Decimal	Char
^@	000	0	NUL
^A	001	1	SOH
^B	002	2	STX
^C	003	3	ETX
^D	004	4	EOT
^E	005	5	ENQ

^F	006	6	ACK
^G	007	7	BEL
^H	010	8	BS
\t	011	9	HT
\n	012	10	LF
^K	013	11	VT
\f	014	12	FF
\r	015	13	CR
^N	016	14	SO
^O	017	15	SI
^P	020	16	SLE
^Q	021	17	SC1
^R	022	18	DC2
^S	023	19	DC3
^T	024	20	DC4
^U	025	21	NAK
^V	026	22	SYN
^W	027	23	ETB
^X	030	24	CAN
^Y	031	25	EM
^Z	032	26	SIB
^[	033	27	ESC
\034	034	28	FS
^]	035	29	GS
^_	037	31	US
	040	32	SP
\"	042	34	"
\\	134	92	\
^^	136	94	^

^?	177	127	DEL

4. If the character is not included in the groups defined in steps 1-3, it is represented as an octal character in the script. These characters are often referred to as high ASCII characters (those with a decimal value greater than 128), and are represented in the script as "\OOO", where OOO is the octal value for the ASCII character.

Handling Winsock application data flow

Frequently, server programs return unique values (for example, a session ID) that vary with each execution of the script and may be vital to the success of subsequent transactions. To create scripts that include these values, you need to substitute the hard-coded values returned by the server with variables. The following original and modified code examples demonstrate this technique.

Original code

In this script, the server sends a session ID in response to a connection by the client. This session ID is required to successfully complete subsequent transactions.

```
/*
* wsk-AdvancedTechniques_original.c
*
* This script contains support for the following
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;
{

/* Declare Variables */

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();
```

```
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

/////////////////////////////////////////////////////
// The session id returned by the server is
// unique to each connection
/////////////////////////////////////////////////////

/* 21bytes: SessionID=jrt90847\r\n */

DO_WSK_Expect(S1, "\n");

//////////////////////////////////////////////////////
// This unique id is then used for subsequent
// requests
//////////////////////////////////////////////////////

/* 34 bytes */

DO_WSK_Send(S1, "SessionID=jrt90847\r\n:^B^@^@^@^B^@^@^@^A^@^@^@");

/* 15 bytes: ID Accepted#^@\r\n */

DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);
}

void abort_function(PLAYER_INFO *s_info)
{

RR__printf("Virtual User %i:ABORTED.", S_task_id);

EXIT();
}
```

Modified code

If the original script (wsk-AdvancedTechniques_original.c shown above) is replayed, it will fail because the session ID will not be unique; rather, it will be the session ID that is coded in the script. To use the unique session ID received from the server, variable substitution must be used.

```
/*
* wsk-AdvancedTechniques_modified.c
*
* This script contains support for the following
* middlewares:
* - Winsock
*/

/* Converted using the following options:
* General:
* Line Split : 80 characters
* Sleep Seconds : 1
* Auto Checkpoints : Yes
*/

#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"

/* set function to call on abort*/

void abort_function(PLAYER_INFO *s_info);
```

```
#ifndef NULL
#define NULL 0
#endif

int rhobot_script(s_info)
PLAYER_INFO *s_info;

{

/* Declare Variables */

char Buffer[64];
char SendBuffer[64];
int nBytesReceived = 0;

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");

// Checkpoints have been included by the convert process

DefaultCheckpointsOn();

DO_WSK_Init(s_info);

SetTimeout(20); /* Wait up to 20 seconds for each expected pattern */

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);

/////////////////////////////////////////////////////
// The reply from the server is read into
// the Buffer variable. We will then have
// the unique Session ID for this connection.
// Also need to null-terminate the buffer
// after receiving.
/////////////////////////////////////////////////////

DO_WSK_Recv(S1, Buffer, 64, 0, &nBytesReceived);
Buffer[nBytesRecieved] = '\0';

/* 21bytes: SessionID=jrt90847\r\n */

//DO_WSK_Expect(S1, "\n");

/////////////////////////////////////////////////////
// Finally, substitute the Session ID received from
// the server with the one coded in the script.
/////////////////////////////////////////////////////

sprintf(SendBuffer, "%s:^B^@^@^@^B^@^@^@^A^@^@^@", Buffer);
DO_WSK_Send(S1, SendBuffer);

/* 34 bytes */

//DO_WSK_Send(S1, "SessionID=jrt90847:^B^@^@^@^B^@^@^@^A^@^@^@");

/* 15 bytes: ID Accepted#^@\r\n */

DO_WSK_Expect(S1, "\n");
DO_WSK_Closesocket(S1);

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);

}
```

```
void abort_function(PLAYER_INFO *s_info)

{
RR__printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
```

Modifying QALoad's functions to incorporate dynamic data

If you need to use dynamic data with your scripts, you can modify some QALoad functions to handle dynamic data. The two scenarios below describe specific situations in which you might need dynamic data, and how to achieve that in the script.

Scenario 1:

One method of accessing dynamic data is by using a datapool file. However, you might need to read in data that is not in the format of an ASCII string, which is required for datapool files.

For example, if the string "\121\101\114\157\141\144" is read in from a datapool file with one of the datapool functions, the output would be "\\121\\101\\114\\157\\141\\144", which is incorrect. To work around this problem, you can use the OctalToChar() command to convert any octal sequences into their binary representation. The following examples illustrates the use of the OctalToChar() command for this purpose:

Example

In this example, the string "\121\101\114\157\141\144" is read in from a central datapool file and converted to its binary representation.

```
/* Declare variables */
char temp[40];

...

BEGIN_TRANSACTION();
GET_DATA();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);

strcpy(temp,VARDATA(1));

OctalToChar(temp); //used to convert octal strings
                   //to their binary format

DO_WSK_Send(S1,temp);
//DO_WSK_Send(S1,"\121\101\122\165\156");
DO_WSK_Closesocket(S1);
```

The DO_WSK_Send() command above sends the string "121\101\114\157\141\144" to the server. This string is the octal representation of the string " QALoad ".

Scenario 2:

You might find that your capture data is not the same data you need for running a test. For example, you might need to change the value of a user ID during replay. One method of changing the value is to change the value through the DO_WSK_Send() command, but that results in the value being static only within the function. To substitute a different value each time, create a dynamic variable, such as a datapool value, to replace the user ID.

Example

In this example, the script includes a DO_WSK_Send() command that sends “name=Jim” to the server as the user ID. Then a variable is used to change the name to “Mark”.

```
/* Declare variables */
char buffer[65];
char sendbuffer[65];

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1,ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//original DO_WSK_Send(S1,"name=Jim");

strcpy( buffer, "Mark");
sprintf( sendbuffer, "name=%s", buffer);
DO_WSK_Send(S1, sendbuffer);

/* 2 bytes: ok */

DO_WSK_Expect(S1,"ok");
DO_WSK_Closesocket(S1);
```

The buffer before the DO_WSK_Send() command is modified and a new buffer is passed as the second parameter of the DO_WSK_Send() command. This effectively sends “name=Mark” to the server instead of "name=Jim”.

Saving server replies

There are two methods for saving the entire reply that a server sends back. The following paragraphs describe each method.

Using the Response() and ResponseLength() commands

The Response() command can be called directly after the DO_WSK_Expect() command. It returns a pointer to the data that has been received by DO_WSK_Expect(). To also receive the length of the replay, call the ResponseLength() command, which returns the number of characters that were received. The following example uses the Response() and ResponseLength() commands.

Example

In this example, variables are declared to store the results from the two functions. Both functions are also used to save the buffer that is received within the DO_WSK_Expect() command.

```
/* Declare Variables */
int x = 0;
char *temp;

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 21 bytes: You are now connected */

DO_WSK_Expect(S1, "d");
```

```
// Used to store the data that was received by the
// DO_WSK_Expect

temp = Response();

// Used to get the size of the response that was received
// so far

x = ResponseLength();

/* The line below will print the length of the response and the actual response */

RR__printf("length = %d, and response= %s",x, temp);
DO_WSK_Closesocket(S1);
```

The message "length=21 response=You are now connected" displays in the Player buffer window.

Using the DO_WSK_Recv() command

To save a response based on its size instead of a unique character string that is used within the DO_WSK_Expect() command, use the DO_WSK_Recv() command. This command enables you to specify how much data to receive and where to store the data.

You can also use the DO_WSK_Recv() command to store the reply that is returned from the server. This strategy is useful when you need to retrieve the buffer that is returned from the server, even though the returned data is too dynamic and causes the DO_WSK_Expect() command to fail every time.

Example

In this example, the DO_WSK_Recv() command is used to save a server reply based on size. Two variables are declared to store the results from the DO_WSK_Recv() command.

```
/* Declare Variables */
int size = 0;
char temp[45];

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 21 bytes: You are now connected */

memset(temp,'\0',45);
DO_WSK_Recv(S1,temp,45,0,&size);
RR__printf("size=%d string=%s",size,temp);
DO_WSK_Closesocket(S1);
```

The message "size=21 string=You are now connected" displays in the Player buffer window.

> Note: If you use this method as a substitute for the DO_WSK_Expect() command, ensure that you receive the correct information prior to calling the next function in the script.

Parsing server replies for values

To parse a buffer for a particular value, you can write a parsing routine that searches the entire buffer for the value. However, you can also use one of QALoad's Winsock helper commands. The following scenarios describe two situations in which you could use the Winsock commands to solve a parsing problem.

Scenario 1:

To find a string in a server reply, you can use the SkipExpr() and ScanExpr() commands. SkipExpr() searches for the first occurrence of a string in the internal buffer that contains the response that was received within the DO_WSK_Expect() command. Then, use the ScanExpr() command to search for another string. ScanExpr() saves the buffer from the first occurrence of the string that was used with SkipExpr() up to and including the string used within ScanExpr(). The first parameter of ScanExpr() is a UNIX-style regular expression. The following table lists the most common expressions:

Character	Meaning
.	Matches the end of a string.
*	Matches any number of characters.
?	Matches any one character.

Example In this example, the buffer contains "sessionid=1234567890abc", and the goal is to retrieve everything after the "=", up to and including "abc".

```
/* Declare Variables */
char temp[35];
int size = 0;

...

BEGIN_TRANSACTION();

...

DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 23 bytes: sessionid=1234567890abc */

DO_WSK_Expect(S1, "c");
SkipExpr("sessionid=");
size=ScanExpr(".*abc" , temp);
RR__printf("length = %d string = %s", size, temp);
DO_WSK_Closesocket(S1);
```

The message "length=13 string=1234567890abc" displays in the Player buffer window.

Scenario 2:

You may have data returned from the server that is too dynamic, that is, you are not able to base parsing on actual characters. The solution is to base the parsing on character positions instead.

For example, to save the characters 20 through 25, you could use the ScanSkip() and ScanString() commands. ScanSkip() skips a specified number of characters in the internal buffer that stores the response that was received within the DO_WSK_Expect() command. ScanString() scans a number of characters from the current position within the buffer into a character string.

Example

In this example, a buffer containing "xxx123456789yyy" is returned from the server. The value between "xxx" and "yyy" is returned.

```
/* Declare Variables */

char temp[15];

...

BEGIN_TRANSACTION();
```

```
...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 16 bytes: xxx0123456789yyy */

memset(temp,'\0',15);
DO_WSK_Expect(S1, "yyy");
ScanSkip(3);
ScanString(10,temp);
RR__printf("string=%s",temp);
DO_WSK_Closesocket(S1);
```

The message “string=0123456789” displays in the Player buffer window.

WWW

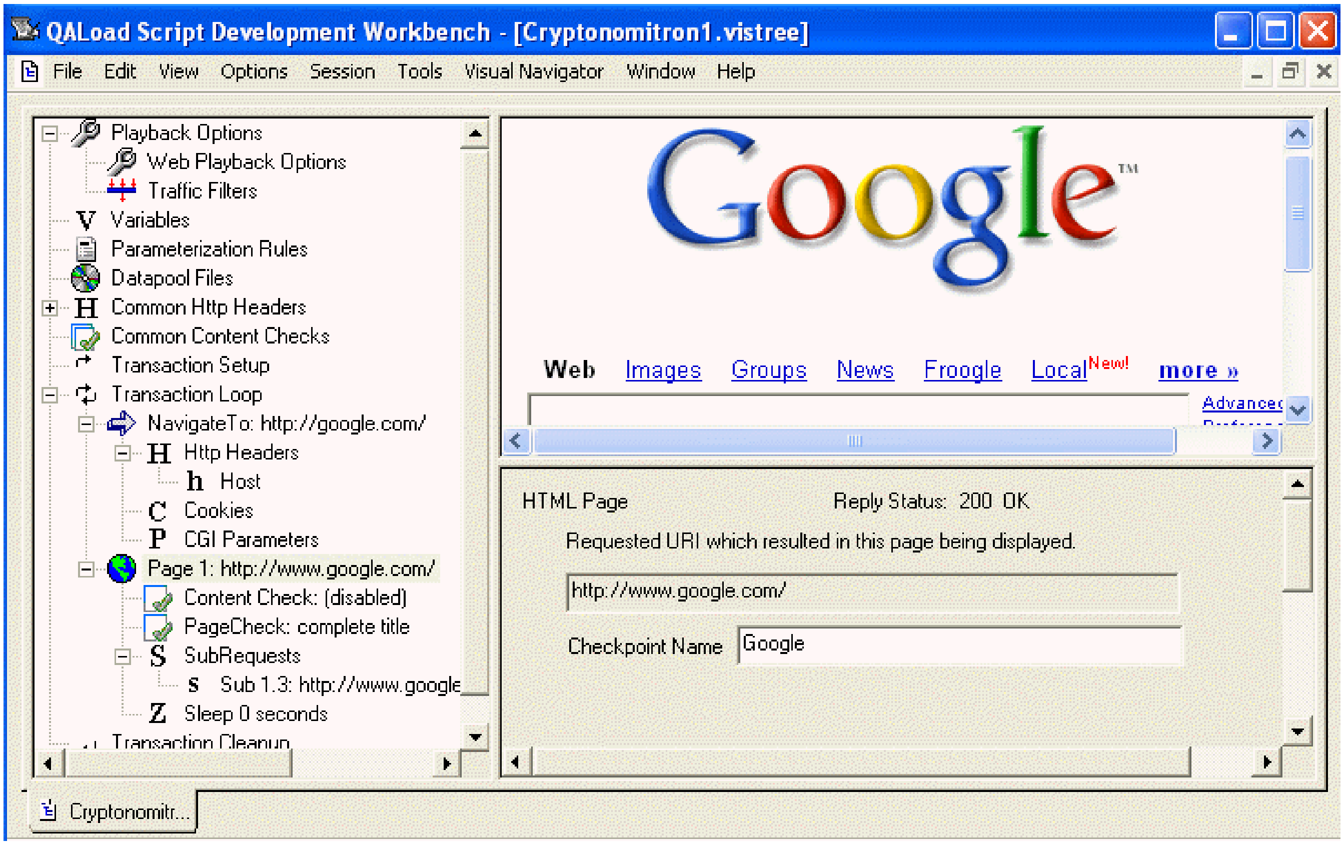
Visual Navigator

The Visual Navigator

The visual scripting interface, called the Visual Navigator, has three panes that represent different aspects of your script, and menu items that offer you additional functionality. Using the Visual Navigator to develop your scripts makes your job easier. For example, searching through lines of code to locate a particular button you clicked on a particular page can take a long time, but using the Visual Navigator you can simply click through the pages to locate that button. In fact, you can develop your whole script – recording, variablizing, converting, compiling, and running it – all from the Visual Navigator's interface without ever writing a line of code.

For a brief explanation of each Visual Navigator pane, click on the panes in the graphic below. For more information about a pane, use the links listed after the graphic.

> Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the View menu.

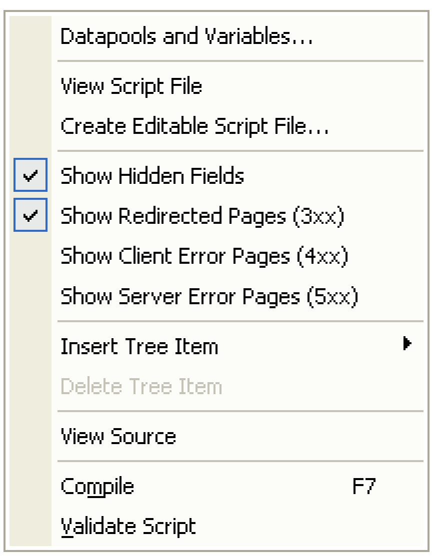


Visual Navigator's menus

The Visual Navigator has a number of special menu commands to help you develop your script.

Visual Navigator menu

The Visual Navigator menu is the main menu for Visual Scripting. Access the Visual Navigator menu from the Script Development Workbench's main menu, or by right-clicking on any item in the Visual Navigator tree-view (left pane).



Datapools and Variables: Opens the Datapools and Variables dialog box, where you can add, delete, or modify datapool files or variables.

View Script File: Opens a window showing the C++ (.cpp) file based on what is currently showing in the Visual Navigator's tree-view. This is a read-only script.

Create Editable Script File: Creates an editable C++ (.cpp) script based on the current Visual Script. You can modify this script directly; however, any changes made to the script will not be reflected in the tree and vise-versa.

Show Hidden Fields: Displays form fields that are hidden by the browser.

Show Redirected Pages (3xx):Toggles whether or not redirected pages are displayed. These are pages that come back with a reply status code of 3xx, for example: `302 Not Found`.

Show Client Error Pages (4xx): Toggles whether or not Client Error pages are displayed. These are pages that come back with a reply status code of 4xx, for example: `407 Proxy Authorization Required`.

Show Server Error Pages (5xx): Toggles whether or not Server Error pages are displayed. These are pages that come back with a reply status code of 5xx, for example: `503 Service Unavailable`.

Insert Tree Item: Opens a sub-menu where you can choose to insert certain tree items into your script. For details, see Inserting script items.

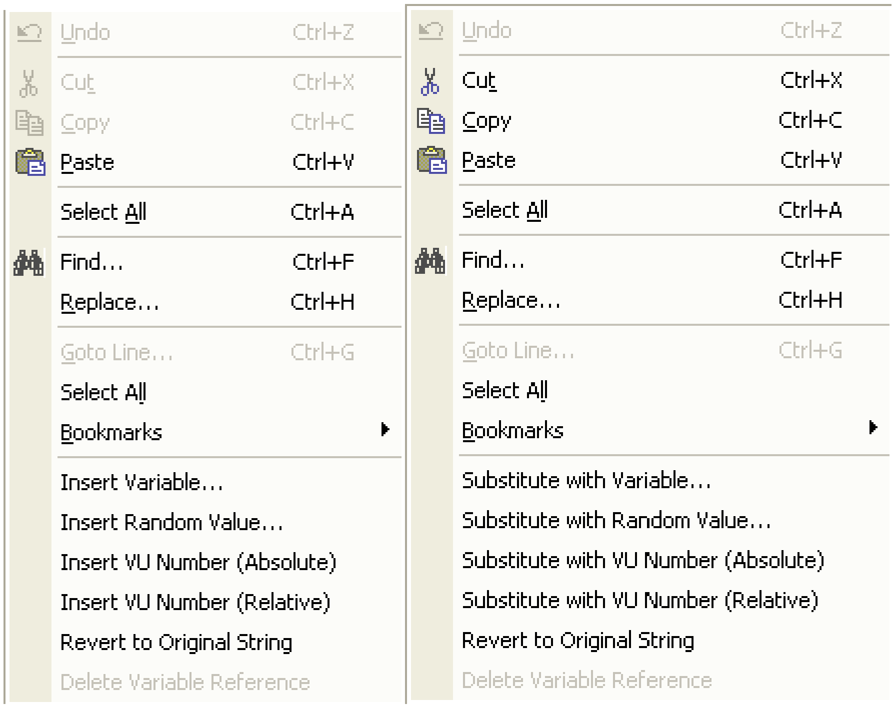
Delete Tree Item: Deletes the currently selected tree item. If the selected item may not be deleted from the script, this command is unavailable.

View Source: Opens a text window displaying the source code of the currently active HTML Page or Subrequest in the tree view.

Edit menu

The Script Development Workbench Edit menu provides special commands for Visual Navigator functionality as well as common Edit menu commands. Access the Edit menu from the main menu, or by right-clicking on an edit box that can be variablized in the Visual Navigator form-view (bottom pane). Fields that can be variablized are denoted with a Var button.

The commands on the Edit menu are dynamic and the availability of certain commands depends upon whether you have text selected and where your cursor is. The following graphics illustrate the difference:



Insert Variable/Substitute with Variable: Opens the Datapools and Variables dialog, allowing you to insert a variable or replace the selected text with a variable. Substituted text will refer to a local variable or datapool variable and will look similar to one of the following examples:

```
{$ VAR:Customer Number $}
```

```
{$ VAR:Last Name:Customer Data $}
```

These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var or Var Wiz next to it).

Insert Random Value/Substitute with Random Value: Opens the Random Number Tag dialog box where you can specify a range within which a random number should be generated for this value. The substituted text will look like this:

```
{$ RANDOM:0:100 $}
```

and it produces a random number between the lower and upper limit each time it is executed.

These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var next to it).

Insert VU Number/Substitute with VU Number (Absolute): Inserts or replaces the highlighted text with the following text:

```
{$ VU:ABS $}
```

This is the virtual user number at runtime. The absolute virtual user number is assigned depending on the number of Players in use. For example, two Player machines with 50 virtual users on each, would assign numbers 0 through 49 for Player 1 and 50 through 99 for Player 2. Typically, the VU number is combined with other text to form a larger string, such as:

```
Customer{$ VU:ABS $}
```

In this example, Player 1 has values of Customer0 through Customer49, and Player 2 has values of Customer50 through Customer99. These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var or Var Wiz next to it).

Insert VU Number/Substitute with VU Number (Relative): Inserts or replaces the highlighted text with the following text:

```
{$ VU:REL $}
```

This is the virtual user number at runtime. The relative virtual user number is assigned to each Player in use. Each Player has relative numbers from 0 to N, where N is the total number of VUs run on that Player. For example, two Player machines with 50 virtual users on each, would assign numbers 0 through 49 to each Player. Typically, the VU number is combined with other text to form a larger string, such as:

```
Customer{$ VU:REL $}
```

In this example, Player 1 has values of Customer0 through Customer49, and Player 2 has values of Customer0 through Customer49. These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var next to it).

Revert to Original String: Rolls the contents of the selected edit box back to when it was first created, usually when the recording was converted to a Visual Script.

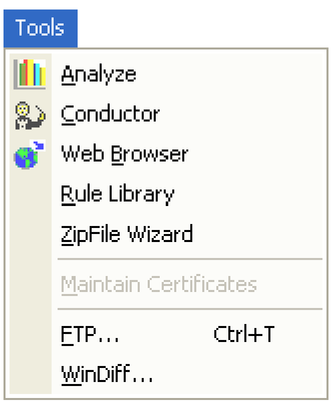
This menu item is enabled only if the edit box within the form can be variablized and it has been changed at some point.

Delete Variable Reference: Deletes the variable from the selected edit box. Note that you can also highlight a variable and press the Delete key to delete a variable within an edit box.

This menu item becomes enabled when you highlight a variable inside of an edit box.

Tools menu

The Script Development Workbench Tools menu provides access to the Rule Library for parameterization, the ZipFile wizard for collecting files needed by the Technical Support team to analyze and resolve a problem, and common Tools menu commands. Access the Tools menu from the Script Development Workbench main menu.



Rule Library: Opens the Rule Library dialog box, where you can add, delete, or modify saved Variable Replacement Rules.

Visual Scripting Concepts

Introducing Visual Scripting

Visual Navigator for WWW is QALoad's easy-to-use visual interface to QALoad's powerful script development tools. Visual Navigator for WWW renders your recorded C++-based transaction in a tri-paned, browser-like environment similar to popular visually-oriented development tools, with icons representing all the elements of your script. In fact, you could set up and run a WWW script without ever having to modify a C++-based script.

With Visual Navigator's advanced editing features, you don't have to know the syntax of QALoad's command set or your HTML requests or responses to customize your script. You can quickly and easily:

- See what URL calls were made and what type they were (for example a POST or GET statement)

- See what information was passed in a call
- See what replies/pages were returned
- Add checkpoints or comments into your script
- Move the begin/end transaction statement
- Move the synchronize statement
- Edit an HTTP header
- Set particular flags and commands
- Add datapools
- Parameterizing your script
- Extract information from a reply to use in subsequent calls
- Save your script and go back to it at any time for further editing
- Create a C++-based script file, if you like

Looking at a transaction loop

The transaction loop is the portion of your script that is played back repeatedly, representing multiple users making requests. The elements in your transaction loop depend on what was originally recorded on each page you requested. You can move the transaction loop up or down in the tree-view using the arrow buttons, to allow certain requests to be moved in or out of the Transaction Setup area, where they will be executed before beginning the transaction loop.

> Note: The following graphic does not show all the possible script elements, but gives a good representation of what your transaction loop might look like in the Visual Navigator.

High-level script items

There are three high-level script items in the transaction loop that represent the web pages you've recorded. NavigateTo, HTML Pages, and XML Requests:

NavigateTo: This is always the first item under the Transaction Loop element, and is always denoted with an arrow icon. It lists the URL that was typed into the web browser at the start of recording. This specifies the first request to be made. The result of this request is the next item in the tree, which is generally an HTML Page item.

If the first item is an HTTP request for XML data, it will appear as an XML Request item in the tree.

Page (HTML): Following NavigateTo there will typically be a set of HTML Page items, which are always denoted with a globe icon underneath the Transaction Loop element. These represent pages visited while the transaction was being recorded.

The form-view (bottom pane) lists the request's reply status, the requested URI, and the associated checkpoint name for the page returned.

HTML Page items can be parent to a number of script items in the tree-view, such as Action items. For more information about sub-items that can exist under a Page item, see HTML Page sub-items.

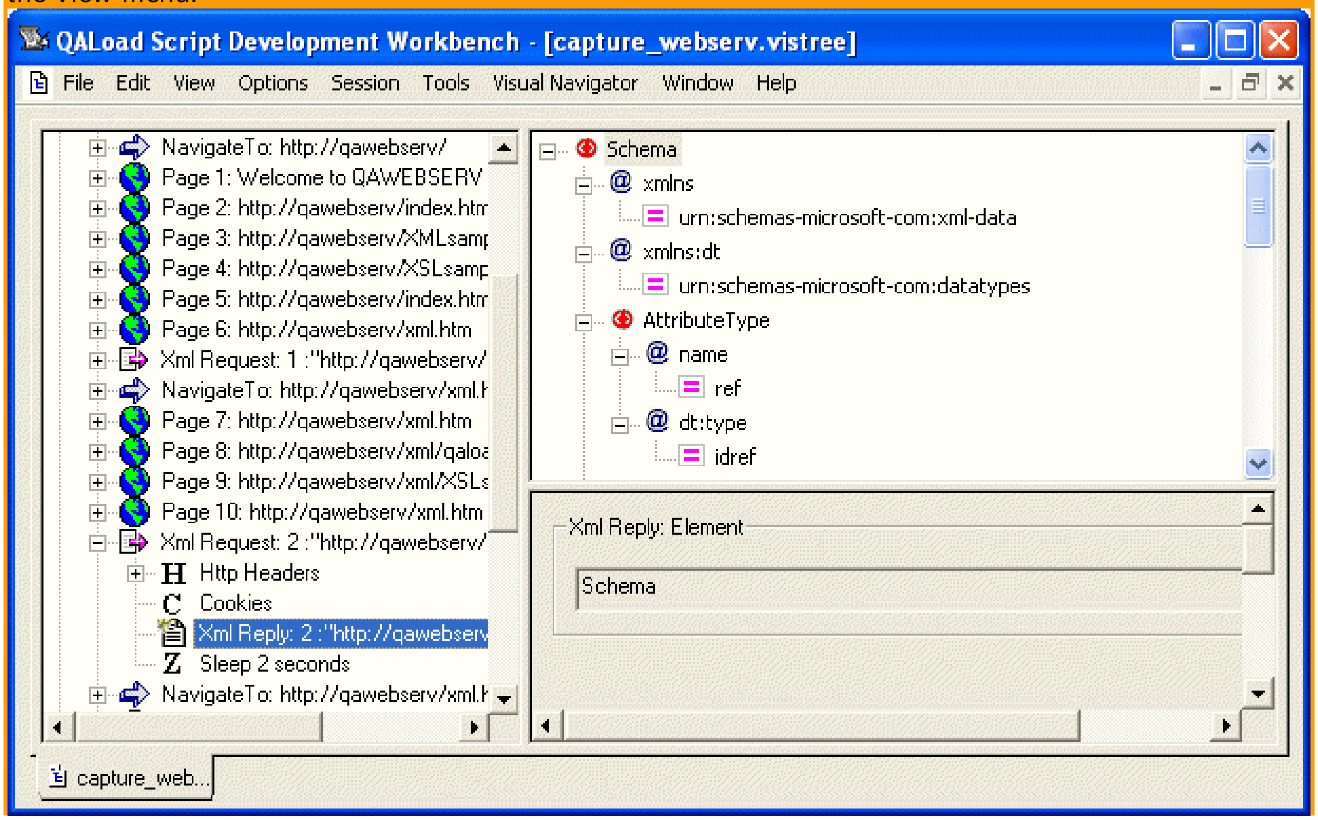
XML Request: Requests for XML documents are denoted by a document/arrow icon underneath the Transaction Loop element. These represent the requests for XML data made during the transaction that was recorded. XML Request items can be parent to a number of lower-level script sub-items in the tree-view, such as Header and Cookie items and the XmlReply document item. See XML requests to learn about sub-items that can exist under an XML Request item.

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use, visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request or an HTTP POST request with an XML document as the post data, the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the View menu.



Streaming media in Visual Navigator

If you selected the Streaming Media option on the WWW Advanced conversion options dialog box before recording your script, and the recorded transaction contains RealOne Player or Windows Media streaming requests, your streaming media request will be presented as a Page in the tree-view, similar to the following graphic:

The form-view (bottom pane) for a streaming media page will show the title Real Media Request or Windows Media Request to indicate the type of request you recorded, and will list the following fields:

Requested URI: Lists the requested URI that invoked the media player. For Real Media the file typically will be an RM file, while for Windows Media it will typically be an ASX file.

Play Media Request: Select this check box for the virtual user to process the RM or ASX file that is received and make the necessary requests to duplicate what the client performed while receiving the streaming media. If this checkbox is not selected, then no further processing is performed after receiving the RM or ASX file.

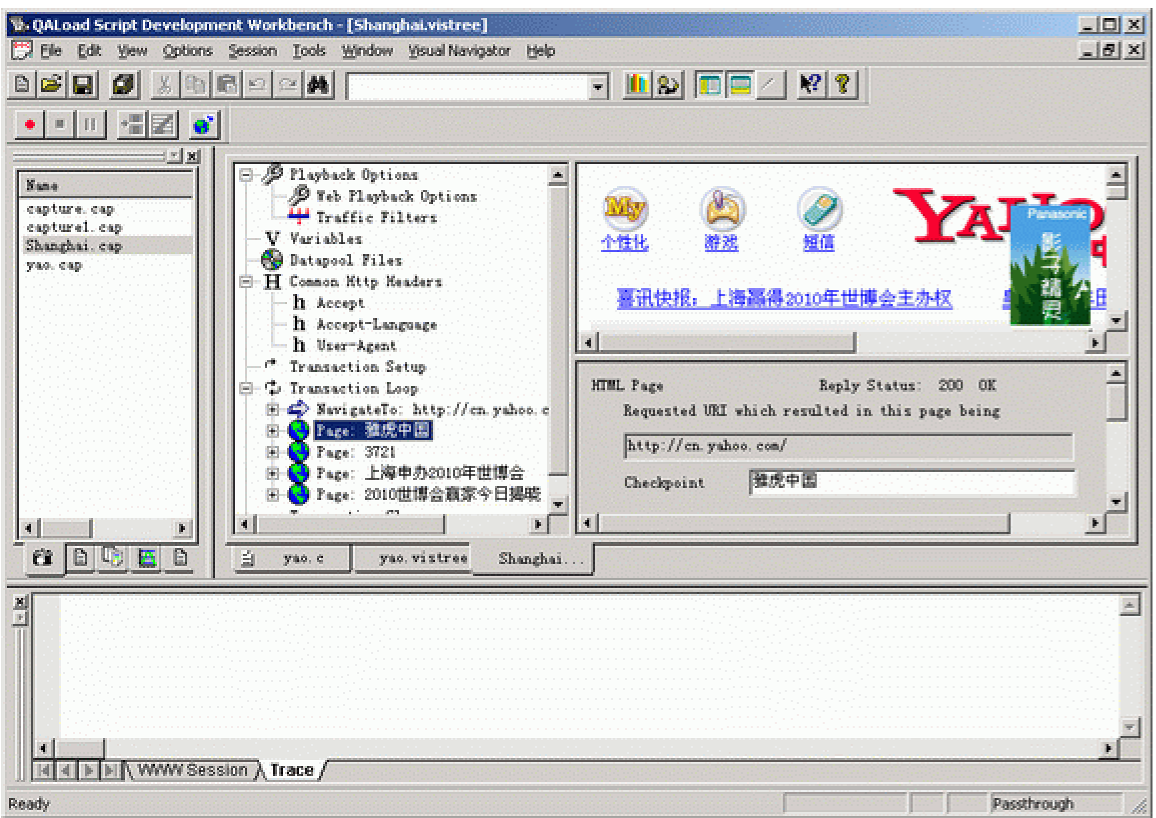
Play Requested Media for N seconds: You can specify how much of the streaming media file the virtual user should play, in seconds, before moving on to the next request. A value of zero indicates that the entire media stream should be played.

> Note: While a virtual user is playing a media request it will not make any other requests in the transaction loop. This may be different than what the user performed when recording the transaction because a browser is capable of spawning the streaming media player as a separate executable which can execute at the same time that the user continues to make further web requests in the browser.

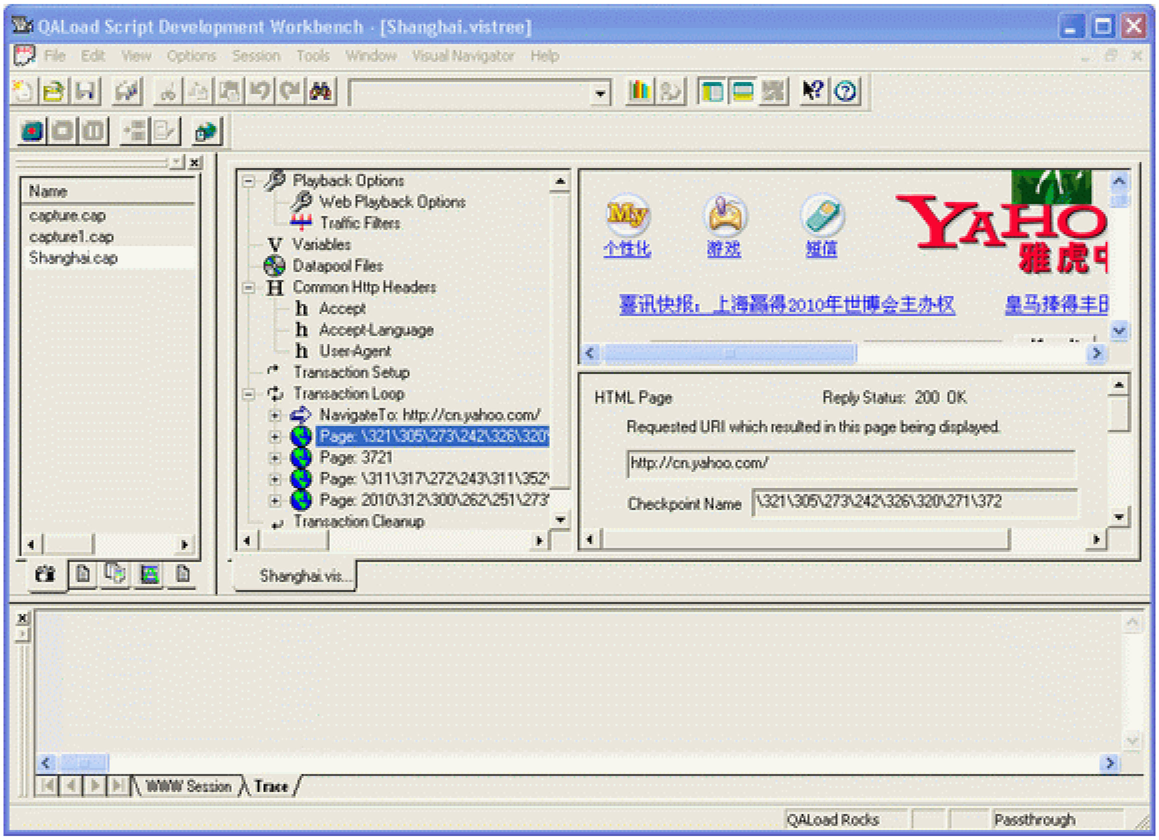
CJK and Visual Navigator

The Visual Navigator handles both native and encoded Chinese, Japanese, and Korean (CJK) characters. (See CJK Support in QALoad for more information about CJK support.)

The following graphic shows how the Visual Navigator provides native character support. Both English and Chinese characters are displayed in the Workbook Pane.



The same capture file, Shanghai.cap, is open in the graphic below. Here, the Visual Navigator displays the Chinese characters in encoded format.



Primary Script Elements

Elements of a Visual Navigator script

When you open a Visual Navigator script, you'll see standard elements of your script listed in the left pane. Each element can contain a number of script items, which in turn have attributes that are editable in some cases. This topic lists the major elements of a script, and links to additional help topics describing each element's associated script items:

The main elements of a Visual Navigator script are:

Playback Options

Web Playback Options

Traffic Filters

Variables

Parameterization Rules

Datapool Files

Common Http Headers

Common Content Checks

Transaction Setup

Transaction Loop

Transaction Cleanup

Playback Options

Lists various settings related to playback. Settings are contained in the Web Playback Options and Traffic Filters items.

Web Playback Options

This item contains settings related to playback such as proxy settings, time out value, number of concurrent connections, baud rate emulation, and filters. This form is divided into three areas: Proxy Settings, Miscellaneous Playback Settings, and Memory Options.

Proxy Settings

This section of the form shows the proxy settings as they were originally recorded. This is the same information you could enter into the Internet Explorer or Netscape browsers to configure them.

Use Auto Configuration Script: If an auto-configuration script was used while recording, then this checkbox will be selected and the address of the auto configuration script will be shown. If an auto configuration script was not used, then the other proxy setting fields are enabled.
HTTP Proxy: The address of the proxy server machine and its port number.
SSL Proxy: The address of the SSL proxy server machine and its port number.
Exceptions: Separate entries with a comma, for example: company.sample.com, company2.company.com

Basic Authorization

Username: A valid user ID to use if the proxy server requires authorization.
Password: A valid password corresponding to the user ID in the previous field, to use if the proxy server requires authorization. The password will be encrypted when recording.

Proxy Http Version: Specifies which protocol to use when making requests through a proxy. Options are 1.0 or 1.1. The version affects whether or not replies may come back chunked. Only HTTP 1.1 requests receives chunked replies.

Miscellaneous Playback Settings

This section contains various settings used for playback. The values on this form initially originate from the WWW Convert Options dialog box; however, you can change some of the values on this form.

Http Version: Specifies which protocol to use when making HTTP and SSL requests during playback. Options are 1.0 or 1.1. The version affects whether or not replies may come back chunked. Only HTTP 1.1 requests receives chunked replies.

Server Response Timeout: The number of seconds to wait for a response from the server before considering it to be timed out.
Max Connection Retries: Specifies how many times a connection will be tried before it is considered failed.
Max Concurrent Connections: The maximum number of concurrent connections that can be opened to request documents. QALoad supports from 1 to 4 connections. This option allows QALoad to better simulate a browser's behavior while playing back a script.
Persistent Connection During Playback: When selected, QALoad will try to maintain an open connection to the server during playback. An open connection may better simulate the recorded transaction.
Caching: When selected, QALoad will simulate the browser's caching behavior.
Reuse SSL Session ID: When selected, QALoad will reuse the current session's communication information (session ID) for all page requests within the transaction.

Use Transmission Baud Rate Emulation: Select to simulate a specific baud rate for transmission of requests, and then select a rate from the drop-down field. QALoad will then slow the transmission of requests appropriately in order to emulate the transaction rate of a modem.
Use Reception Baud Rate Emulation: Select to simulate a specific baud rate for reception of requests, and then select a rate from the drop-down field. QALoad will then slow the reception of requests appropriately in order to emulate the transaction rate of a modem.
Refresh Timeout (Seconds): A page will be considered a redirect if the META Refresh CONTENT field's value is less than the value in this field. This field can only be set in the conversion options.

Memory Options

This section contains sections related to memory usage at the virtual user level.

Virtual User Memory Size: Indicates whether the current setting is for Typical or Minimize Memory Usage.

Memory Options: Click to open the Memory Options dialog box, from which you can change from Minimize Memory Usage mode to Typical mode, or change Typical mode settings. Once a script is converted using Typical mode, the memory usage setting cannot be changed to Minimize Memory Usage.

Manually Select Subrequests: Indicates whether the current setting is Active or Not Active. This option can be modified on the Memory Options dialog box. When the option is Active, the following buttons are also available.

Select All Subs: Click to select all subrequests.

Clear All Subs: Click to clear all subrequests.

Traffic Filters

Traffic Filters allows you to filter out certain requests while playing back a script. For example, you may want to eliminate advertisement requests that are being made of Web servers other than the one you are testing.

The first time you convert a capture file, it will use the filters specified on the WWW Advanced conversion options dialog box and will eliminate any pages indicated as being filtered. Filter information entered on that dialog will be written to the Traffic Filters tree item. Any changes you make to this tree item will not affect items in the Visual Navigator tree but will be strictly for filtering subrequests at playback time.

Note: Traffic filters do not affect which XML requests are written to the tree when a capture file is converted.

Two types of filters may be specified:

- **Allowed Traffic**

This list specifies a set of substrings for the type of URLs you will allow to be requested. The URL of every request made must include at least one of the specified substrings. The filter only applies to subrequests.

For example, if the list contained the strings "compuware" and "compuweb" then QALoad would allow the URLs "www.compuware.com\info.htm" and "http://mycompuweb/weather.htm" to be requested, but not "www.google.com".

- **Blocked Traffic**

This list specifies a set of substrings for the type of URLs which should not be requested. If a URL to be requested contains any of the substrings in this list, then the request will not be sent. The filter only applies to subrequests.

For example, if this list contained the terms "doubleclick" and "popup_source" then QALoad would not allow the following URL to be requested: "http://www.doubleclick.com/myAdvertisement.htm".

The traffic filters form-view also lists the substrings that were converted to Additional Subrequests during conversion under the Traffic converted to Subrequests heading. This list is read-only and displays the settings from the WWW Advanced conversion options dialog box.

Variables

Lists local variables that have been created for this script. For information about parameterization, see About Parameterization.

Form View Fields

When you highlight a variable in the tree view, the form view displays fields related to the variable type.

Variable Name: The name you provided to identify the variable.

Initial: The initial value for this field, before parameterization takes place.

Note: Variables that are saved in the Parameterization Rule Library can be changed only from the Rule Library.

Datapool Files

Lists datapool files being called by the script. Each datapool listed has a list of variables under it representing columns in the datapool file. Datapools can be Local (specific to a single Player) or Central (available to all Players).

Common Http Headers

Lists headers that were recorded from at least 50% of your requests. These headers will be sent out with every request that is made at playback unless they are overwritten by a header of the same name underneath an individual request action.

You can insert new header items from the tree-view by clicking Visual Navigator>Insert Tree Item>Http Header. In addition, you can modify the values in the Http Header form in the right pane.

Common Content Checks

Lists common content checks, which apply to all replies sent by the server. Content checks enable you to verify whether the correct page was returned based on the existence or absence of a specific search string. You can also set content checks at the page level. Click the Add New Content Check Item button in the form-view to add new common content checks.

Common content checks can include variables. Common content checks enable you to generate an error code on a set condition even if no individual page-level content checks are enabled. The search string is compared to the raw HTML returned by the server, so you may need to include HTML tags in your search to match the text that appears in the browser.

Transaction Setup

Lists any actions that occurred before the main transaction loop. Any items/actions that occur under this heading will be executed after the Synchronize but before the BEGIN_TRANSACTION(); statement at playback. For example, you may have logged in to a particular Web site and do not want to log in and out with every transaction at playback. You can move the Transaction Setup item in the tree-view by highlighting it and clicking the Move UP/Move DOWN buttons. The Transaction Setup can contain client certificate tree items.

Client Certificate tree item

If the recorded transaction contains a line with a ssl-clientcert command, then Visual Navigator will create a Client Certificate tree item and place it directly beneath the Transaction Setup tree item.

The Client Certificate string can be modified or variablized in the form-view.

The Client Certificate item can also be moved up and down the tree like other tree items, such as checkpoints. This allows you to move it into the Transaction Loop area if you wish to change the certificate with each transaction.

A Client Certificate item will generate a script line similar to the following:

```
Set (EVERY_REQUEST, CERTIFICATE, “qaload_cl”);
```

If the Requires Password check box is selected, the generated script line is similar to the following:

```
Set (EVERY_REQUEST, CERTIFICATE_PASSWORD, "~encr~250F7641455876");
```

Transaction Loop

Lists the requests in your transaction. All items/actions that occur under this heading are placed between the BEGIN_TRANSACTION and END_TRANSACTION statements, causing them to be repeated for as many times as the Conductor tells them to be. The transaction loop has a number of possible sub-elements, depending on the Web site you tested. For detailed descriptions of the elements that can be listed in a transaction loop, see Looking at a transaction loop.

Transaction Cleanup

Lists actions that occur after the script has finished executing the appropriate number of transactions. Any items that occur under this heading will be placed after the END_TRANSACTION statement. For example, you may want to log out of a particular Web site after completing the appropriate number of transactions. You can move the Transaction Cleanup item in the tree-view by highlighting it and clicking the Move UP/Move DOWN buttons.

Parameterization Rules

Lists Parameterization Rules that you create when you define a variable to substitute for a value. Saving the variable as a rule enables you to reuse it for other instances of the variable.

When you click Parameterization Rules in the Visual Navigator tree-view, details of the rules stored in the Rule Library appears in the right-hand pane.

Notes:

- Properties for the rules are defined using the Rule Library Wizard or the Variable Replacement Wizard.
- An exclamation mark (!) in the first column, signifies that the rule matches the same parameter as another rule defined in the script. The rule applied first prevents the other rules from being applied. Rules are applied in the order they are displayed in the table.

Rule Name: Name of the rule.

Matching Item: Name of the item within the script that matches the parameters defined in the rule and that will be replaced with the rule.

Matching Type: The type of item within a script that is replaced by this rule. Item types include:

- CGI Parameter
- Cookie
- Header
- EditBox
- RadioButton
- TextBox

Fields Replaced: Fields of the matching item that are replaced by this rule. Fields may be:

- Name
- Value
- Both name and value

Apply: Apply the selected rule.

Apply All: Apply the rule to all matching items in the script.

Undo Apply: Remove the rule from matching items in the script.

Edit: Opens the Rule Library Wizard so you can edit the rule.

Duplicate: Duplicates the selected rule in the list.

Delete: Deletes the rule from the script. The rule remains in the library.

Add to Library: Adds the rule to the Rule Library. Only rules in the Rule Library can be applied to future scripts.

Note: You can create a rule using the Variable Replacement Wizard without adding it to the library.

Goto Rule: Displays details for the selected rule.

Get from Library: Opens the Get Rule from Library dialog box. Select a rule to add to the visual tree.

Rules Details

When you click an individual rule in the Parameterization Rule item in the Visual Navigator tree-view, the form-view displays information for that rule in three tabs.

Note: Properties for the rules are defined using the Rule Library Wizard or the Variable Replacement Wizard.

Apply Rule: Applies the rule to all items in the Items to Replace table.

Undo Apply: Removes the rule from all items in the Items to Replace table to which it has been applied.

Edit: Opens the Rule Library Wizard so you can edit the rule.

Add to Library: Adds the rule to the Rule Library. Only rules in the Rule Library can be applied to future scripts.

Note: You can create a rule using the Variable Replacement Wizard without adding it to the library.

General Tab

Details Section:

Rule Name: Displays the name of the rule. You cannot edit this field.

Description: Displays the description of the rule. This is generated by the Variable Replacement Wizard or supplied by the user.

Status: Displays whether the rule is applied to items in the script.

Folder: Displays the name of the folder where the rule is located.

Last Modified: Displays the date the rule was last modified.

Rule Summary Section:

Gives a descriptive summary of the parameters established for this rule. The rule parameters are defined using the New Library Rule Wizard or the Variable Replacement Wizard.

Matching Items Tab

Matching Item Properties

Item Name: Displays the name by which the parameter is identified in the script.

Example: Shows an example of the parameter.

Item Type: The type of item within the script that is matched by this rule

Host URL: Identifies the URL where the parameters reside that are replaced by this rule.

Items to Replace

Note: When a different rule has been applied to an individual item, an red exclamation mark (!) appears next to the item in the table. Only one rule can be applied at a time. Click Go to Item and review the information in the Matching Parameter Rules area of the form-view to view the rule applied to the item.

Name: Displays the name by which the parameter is identified in the script.

Type: Displays the type of variable in which it is used. This can be:

- CGI Parameter
- Cookie
- Editbox
- TextArea
- Radio button
- Header

Found In: The name of Host URL where this parameter is found.

Apply to Item: Applies the rule to the matching items in the Items to Replace table that you have selected.

Undo Apply: Removes the rule from the matching items you have selected in the table to which it has been applied.

Edit: Opens the Variable Replacement Wizard, where you can make changes to the variable and save it again to the rule.

Go to Item: Takes you to the individual occurrence of the item in the script.

Variables Tab

Name field variable

This area displays when you select Name as the field to replace when you created the rule.

Variable Name: Displays the name of the variable to which this rule applies.

Frequency: Shows the frequency with which the rule is applied to the script.

Calculation: Shows the calculation used for the variable.

Example: Shows an example of the value returned from the calculation.

Details: Displays the Calculation Details dialog box.

Value field variable

This area displays when you selected Value as the field to replace when you created the rule.

Variable Name: Displays the name of the variable to which this rule applies.

Frequency: Shows the frequency with which the rule is applied to the script. This can be : Once per test, Once per transaction, or Every usage.

Calculation: Shows the calculation used for the variable.

Example: Shows an example of the value returned from the calculation.

Details: Displays the Calculation Details dialog box.

Transaction Loop Items

Synch

Inserts a Synch item immediately after the currently selected HTML Page. A Synch item represents a spot where all virtual users will pause during replay until all active virtual users have reached the same point. Once the virtual users are synchronized this way, the Conductor will instruct them all to continue.

A Synch item can be moved up or down the tree using Up/Down in the form-view.

IP Spoof

Inserts an IP Spoof item immediately after the currently selected HTML page.

In order for IP Spoofing to work with Visual Navigator, it is necessary to create or insert an existing local datapool file called QALOAD_IPSPOOF in the Visual Navigator tree-view. For more information about creating this datapool file and inserting it, see Creating the QALOAD_IPSPOOF datapool file.

Read Datapool

Opens the Datapool and Variables dialog box, allowing you to choose which datapool to use, and then inserts a Read Datapool item immediately after the currently selected HTML Page.

You can move this item up or down the tree-view by clicking Up/Down in the form-view.

Form View Fields

The form-view (bottom pane) for a Read Datapool item displays the following information:

Datapool Name: The name of the datapool file that will be used. To use a different file, click Select file... and then choose or create another.

Datapool Variables: Lists the variables (fields) associated with this datapool file.

Checkpoint pair

Inserts a Begin Checkpoint item before the currently selected HTML Page and an End Checkpoint after the currently selected HTML Page.

Checkpoints are used to measure duration times for certain actions to be completed. You can move either the Begin or End checkpoint item to encompass several requests, if necessary. To move either item, highlight it and then click Move Up/Move Down in the form-view.

Form View Fields

The form-view fields for a Begin/End Checkpoint item lists the following information:

Name: The unique name of the checkpoint. When a checkpoint item is inserted, it is given a default name such as `UserCheckpoint 1`. This name can be changed. Changing the name of the Begin or End Checkpoint item automatically changes the name of the corresponding Begin or End item. In addition, deleting a Begin or End Checkpoint item automatically deletes the corresponding Begin or End item.

Show End/Begin Checkpoint: Click this button to quickly locate the Begin or End Checkpoint item that corresponds to the item currently selected.

Increment Variable/Decrement Variable/Reset Variable

Increments, decrements, or resets the value of a local variable.

Opens the Datapools and Variables dialog box, which allows you to select which variable to increment, decrement, or reset. It then inserts the appropriate item (Increment Variable, Decrement Variable, Reset Variable) after the currently selected HTML page.

Form View Fields

The form-view (bottom pane) for an Increment, Decrement, or Reset Variable item lists the following information:

Variable Name: The name of the variable being modified. To use a different variable, click Select Var… and then choose or create a different local variable.

Action: Describes the type of action to perform on the local variable. Options are:

- **Increment** — increments the value by 1.
- **Decrement** — decrements the value by 1.
- **Reset Value** — Type a value to replace the current variable with in the **New Value** field.

Print Values (debugging)

Inserts a Debug Print item after the currently selected HTML Page. This causes a string to be output to the Player window during playback. This can be useful for debugging a script while you are trying to variablize it so that it replays correctly with multiple virtual users.

Form View Field

Debug Print Statement: If you simply type a string in this field, it is printed in the Player's output window just as you typed it. If you insert a variable into the string, the value of the variable is printed to the Player's window at playback so you can see its value. For example, to print the value of a datapool variable you might enter a string like the following:

```
Customer Name is {$ First Name:Customer Data $} {$ Last Name:Customer Data $}
```

This prints a statement like the following to the Player window at runtime: `Customer Name is Cindy Nelson.`

You can move the statement within the tree by clicking Move Up/Move/Down in the form-view.

Comment

Inserts a Comment item after the currently selected HTML page. Type your comment into the form-view (bottom pane).

HTML Pages

HTML Page form-view

The form-view (bottom pane) for an HTML Page tree item contains the following information:

Reply Status: The code designating the status of the reply. For most pages that were returned correctly, this will be 200 OK.

Requested URI: This read-only field lists the URI which was requested that resulted in this page being displayed.

Checkpoint Name: If the page has a title, then it will be used as the checkpoint name. If not, the word Checkpoint will be used. To make sure all checkpoint names are unique, a number may be appended to the end of the checkpoint name.

Meta Refresh Required [ ] Seconds Before Redirection: If the Refresh Timeout option was selected on the WWW Conversion Options dialog box, this field displays the number of seconds that QALoad waits before it treats a META refresh request as a normal request. This field only appears when refresh timeouts are enabled.

HTML Page sub-items

The following script items can exist under a Page (HTML) item in the Visual Navigator's tree-view. Each possible page sub-item is listed below, along with descriptions for the fields that appear in the form-view in the right pane when you select the item in the tree-view.

In addition, a Page item can contain sub-items that you insert manually after recording the transaction.

Content Check sub-item

PageCheck sub-item

AdditionalSubRequests sub-item

SubRequests sub-item

Cookies Set by Server sub-item

Sleep sub-item

Fill In Form sub-item

Action sub-items

Content Check sub-item

Inserts a Content Check item for the currently-selected HTML page. This verifies that the correct page was returned based on the existence or absence of a particular search string in the server's reply for that page. Content checks can include variables. The search string is compared to the raw HTML returned by the server, so you may need to include HTML tags in your search to match the text that appears in the browser.

The top pane displays the source for the HTML page. You can easily select text in the top pane and add it to the content check definition by clicking the Copy from Source button.

Form-view fields

The form-view (bottom pane) includes the following fields:

Enable this content check: Select to enable the content check for the page. Content checks are disabled by default. After checking this box, choose whether the check succeeds if the string is contained or not contained in the page. Then type the string text in the box.

String: Enter the string that defines the content check. You can type a string or click the Var... button to select variables for the string.

Var...: Opens the Select Variable dialog box where you can select the variable to add to the string.

Copy from Source: Select text in the top pane and click this button to copy the selected text in the top pane to the text box.

Verify: Click to test whether the check succeeds for the text shown in the top pane.

PageCheck sub-item

Allows you to verify that the title of the page that was requested is correct.

Form View Fields

The form-view (bottom pane) includes the following fields:

Original Title: This read-only field displays the title of the recorded page.
Verify Page Title: Select this checkbox to enable/disable title verification at playback, then select the appropriate option:
Complete Title: The title of the page that is requested must match exactly with the original title.
Prefix: Matches the first characters of the title. Specify how many characters to match. At playback time QALoad will then compare the first N characters to the original title, allowing you to verify titles which might change slightly.
Suffix: Matches the last characters of the title. Specify how many characters to match. At playback time QALoad will then compare the last N characters to the original title, allowing you to verify titles which might change slightly.

AdditionalSubRequests sub-item

Some requests are contained in applets, ActiveX components, or other objects that are captured, but not played back by QALoad . These subrequests, which are not recognized as normal subrequests, are listed in the AdditionalSubRequests tree item.

Each additional subrequest item appears in the script as a pre-loaded subrequest just before the main action. As a result, the playback engine requests the main page, regular subrequests, and then the pre-loaded subrequests.

For example:

```
//---------  REQUEST # 2  ---------
//
//  current page url is http://c96852d01/pda/
//
// Pre-load the following image requests before the next request is made.
// These requests seem to have been made by javascript or applets associated
// with the next page but will not be made automatically by the replay engine,
// hence they are here in the script.
//

Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,
      "http://c96852d01/pda/images/LeftBackgrnd.jpg");

Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,
      "http://c96852d01/pda/menuopen.gif");
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,
     "http://c96852d01/pda/menuclose.gif");
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,
     "http://c96852d01/pda/menuclose.gif");
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,
     "http://c96852d01/pda/images/browsex.gif");

Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://c96852d01/pda/images/browse.gif");
```

SubRequests sub-item

Lists all subrequests (such as images) that the page performed in order to be fully rendered in the browser. Subrequests cannot be changed and are shown strictly to provide detailed information about the requests that were made during the recording session.

Form View Fields

The form-view (bottom pane) for subrequests contains the following fields:

URI: Lists the URI of the subrequest that was made.
View Source: Click to open the source file associated with the reply of the subrequest in a text format. Image files will show up as mostly unreadable characters, but cascading stylesheet and JavaScript requests will be readable.

Cookies set by Server sub-item

If the reply from the server for the requested page contains a Set-Cookie command, it is listed here. This item cannot be modified, it is listed for your information only.

Form View Fields

The form-view (bottom pane) for cookies lists the following information:

Name: The name of the cookie.
Value: The value that the cookie should be set to.
Path: The path within the domain to which this cookie is assigned. This cookie will only be sent to the server if future request URIs have the same domain and this path. The path can be deeper but it must start with this path.
Domain: The domain to which this cookie is assigned. This cookie will only be sent to the server if future request URIs have this domain and the path specified above.
Expires: The cookie's expiration date, if it has one.

Sleep sub-item

Every page will have a Sleep item immediately before its Action item. The sleep value specifies how many seconds were spent viewing this page (or filling out a form) before an action was taken (such as clicking on a link or button).

Fill In Form sub-item

If a requested page contains a Form (html element) that was filled in by the user, then a Fill In Form item and its associated elements will be created in the tree-view. When this tree item is selected, a blinking frame will appear around the form in the browser-view (top pane).

Form View Fields

The form-view (bottom pane) for a Form element lists the following information:

Form Action String: The Action String associated with this form. For example, the URI which will be requested.
Form Number: The number assigned to the form, indicating which form will be used.

A Fill In Form item in the tree-view can include a number of sub elements. For details, see Form sub-items.

Extract String sub-item

Inserts an Extract String item when you need to extract information from the script and store it in a variable to use later in the script.

When an Extract String item is inserted into the tree-view, the browser-view displays the HTML source for the page in which the item is inserted. The string to extract is recognized by the text preceding it and the text following it. The test string in between is extracted and saved into a local variable. You must specify the local variable that receives the extracted string at run time by clicking on Select Var.

Form View Fields

The form-view (bottom pane) for an Extract String item lists the following information:

Search for: Type the text you want to extract.

Find prev: Identifies the previous occurrence of the selected text in the source window (upper pane).

Find next: Identifies the next occurrence of the selected text in the source window (upper pane).

Match Case: Enables you to make your search case-sensitive.

Variable to receive string: The local variable that receives the string extracted from the reply. The initial value is None.

Select Var: Click to access the Datapools and Variables dialog box where you can select or create a local variable to use with the Extract String item.

Preceded by: The text preceding the string you want to extract. This is filled in automatically when you select the text to extract and click Copy from Source. Use the spin controls to increase or decrease the size of the string.

Extracted: The string to be extracted on this particular reply. This is filled in automatically when you select the text to extract and click Copy from Source.

Followed by: The text following the string to be extracted. This is filled in automatically when you select the text to extract and click Copy from Source. Use the spin controls to increase or decrease the size of the string.

Copy from Source: Copies the selected text from the source window and places it in the edit boxes.

Enter String Manually: This option allows you to cut and paste the text you select in the source window into the Preceded by, Extract, and Followed by edit boxes.

Frames

If an HTML page that is recorded contained frames, they will be represented in the tree-view (left pane) with a circle icon containing a capital F. A frame page will be indented beneath the page that is its parent. If you click on a frame icon in the tree-view, the corresponding frame will be highlighted in the browser-view (top pane) with a blinking frame around it for identification.

Duplicated frameset pages

Sometimes when a user clicks on a link or takes some other action inside of a frame, the new page that was requested simply replaces the contents of one of the frames already shown in the browser. To indicate that the frameset page (the main page that holds the frames) has not changed, Visual Navigator renames it Duplicated Frameset n. Where n is an identifying number for the frameset that is incremented as more frameset pages are duplicated.

Form view fields

The form-view (bottom pane) for a Frame Page item lists the following information:

Requested URI: The URI which was used to request the frame page
Frame Name: The name of the frame as indicated in the source HTML.

Action Sub-Items

Action sub-items

An Action item appears under each HTML Page except the last one in the script. This represents the action that the user took to get to the next page. Action items include:

- NavigateTo – Visual Navigator could not determine how the user accessed the next page (they may have typed a URL directly into the address bar, or a JavaScript may have caused the jump).

  Note: A NavigateTo item is the first element that appears under the Transaction Loop element. This is because when the browser is launched during recording, the user must specify a starting address (typically by typing it into the Address bar).
- Click On Link – The user clicked on a text link
- Click On Image – The user clicked on an image link
- Click On Button – The user clicked on a Submit type button.
- PostTo – Data was sent to the server with a POST command but a matching submit button was not found. This may have been caused by a JavaScript.

Action items can contain various sub-elements. For details, see Action item sub-elements.

NavigateTo sub-item

Specifies a URI to be requested. If the Script Development Workbench cannot determine how the next page was requested (typically due to a JavaScript making the request) then it will use a NavigateTo tree item instead of something more specific such as a Click On Link.

Form View Field

The form-view (bottom pane) for a NavigateTo item lists the following information:

URI: The URI that was requested. This value can be variablized.
Use Link Encoding for CGI Parameters: If an HTTP GET request is the result of a direct request that includes the CGI parameters within the link, the parameters must be sent to the server in a different encoding called Link Encoding. In most cases, the Script Development Workbench automatically detects this type of request and selects this option. However, you can manually change this setting for troubleshooting. Generally, If a Navigate_To is associated with a form, use form encoding (do not select this check box); if a Navigate_To is associated with an anchor, use link encoding (select this check box).

Click On Link sub-item

If the user clicked on a text link or an image link, then a Click On Link action item will be inserted under that page. This is used to describe the action that was taken while on this page that resulted in the next page being requested.

When a Click On Link tree-view item is selected, the text or image in the browser-view (top pane) will be highlighted by a blinking frame to make it easier to locate. There are several types of Click On Links:

- Text Links – One of the more common links in web pages are Text based links. These usually appear as underlined text.
- Image Links – An image can have a link, similar to text.
- Client-Side Image Map – A Client-Side Image Map is an image on a page that has multiple links associated with it. Each link is associated with a region, which can be any shape. When the user clicks on the image, the browser will determine which region was clicked on and will request the page linked to from that region.
- Server-Side Image Map – A Server-Side Image Map is an image on a page that has multiple links associated with it. Unlike Client-Side Image Maps, these links are stored on the server rather than the client. When a user clicks on the image, the browser will send the server the mouse coordinate relative to the top-left corner of the image. The server will then reply with the appropriate page.

Form View Fields

The form-view (bottom pane) for Click On Link items list the following information:

Link Type: Specifies the type of link found. For example, Text, Image, Client Side Image Map, or Server Side Image Map.
Link Text: Only available for text links. Shows the text associated with the link, as displayed in the browser-view (top pane). During playback, the script will search for a link using this text and then request the page specified by the link.
Image: Only available for image links. Shows the URL of the image that was clicked. When playing back the script, QALoad will search for an image with a link that originated from this URL and will request the page specified by the link.
If multiple matches to Target URI are found, use match: It could be possible to have multiple links to a single URI. Use the spin control to move to the next or previous link that matches the target URI. Selected links will be framed and the Link Text, or Image, or Image Region edit box (the edit box available depends on what type of link it is) will be updated .
Target URI: Displays the URI that will be requested when this link is clicked on.
X and Y Coordinates: Server Side Image Maps only. The mouse coordinates, relative to the upper left of the image, that were clicked on while recording.

Click On Button sub-item

Clicking on Submit is usually associated with entering values into a form (Fill In Form item). When the Click On Button tree-view item is selected, the associated button in the browser window (top pane) will be highlighted with a blinking frame, making it easier to locate.

Form View Fields

The form-view (bottom pane) for Click On Buttons (submits) lists the following information:

Identify by: Click the appropriate button to choose whether to identify the button by Name (internal name as indicated in the HTML), by Label (the text on the button), by Occurrence (e.g. the 3rd button), or by Image Source (in case this submit button is really an image). The edit Identify by buttons will change appropriately, depending upon which option you choose.

Prev Button/Next Button: If there are other submit buttons on the page, use Prev Button/Next Button to choose to use one of those other submit buttons. The name, label, or occurrence will be updated automatically.

Label: The Label field (or Name, Occurrence, or Image Source field) can be modified or variablized. If you variablize the field, clicking Next Button or Prev Button will not affect its contents. However if you choose to use one of the other options, different text will be displayed because variablization is specific to each method of identification.

If multiple matches to URL are found, use match: If QALoad finds multiple matches (for example, two buttons labeled Buy Me) then you can use this field to specify which match to use. For example you could specify that you want to click on the 4th button with a label of Buy Me when looking at a page with a lot of items for sale.

Original Label: The original label associated with the button that was clicked while recording.

Original Name: The original name (from the HTML source) for this button.

Form Action: The URL associated with a form. This is the address for the request that will sent when the submit button is clicked.

Submit Method: Whether this submit item was a GET or POST.

PostTo sub-item

If the recorded request was a POST request rather than a GET request and the Script Development Workbench could not find a matching Submit type button, then a PostTo tree action item will be inserted under the page. This can sometimes happen if the request is initiated by JavaScript.

Form View Fields

The form-view (bottom pane) for a PostTo item lists the following information:

URI: The URI which is being posted to.
Content Type: One of the following: DEFAULT, TEXT_PLAIN, or MULTIPART_FORM_DATA. This field is read-only.

Action Item Sub-Items

Http Headers sub-item

If a header exists under an Action item, then it will be sent for that request only. If the header has the same name as one of the common headers, then it will override the common header for this request only. It is possible to insert additional HTTP headers.

Form View Fields

The form-view (bottom pane) for an HTTP Header lists the following information:

General

Name: The name of the HTTP header.
Value: The value of the HTTP header.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information on the rule displays in the Matching Item tab of the Rule Details dialog box.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

Cookies sub-item

When a Cookie item is a sub-element of an Action item, it contains a list of cookie items that were sent in the header of the request that the Action item made when recording. Cookies are added automatically by the browser based on the URI that is being requested. They are either set as a result of the previous reply (the server returned a Set-Cookie command), or they are set by JavaScript contained in the previous reply.

If the Cookie shown has a matching Set-Cookie item, then nothing will display in the script since the cookie is created automatically during playback. If there is no matching Set-Cookie item, then a Set-Cookie type statement will be generated in the script.

You can insert additional cookies into the Cookies section of a page as another means of variablizing the playback. How?

Form View Fields

The form-view (bottom pane) for a cookie item lists the following information. Both of these items are can be edited and variablized. You can also insert additional cookie items.

General

Name: The name of the cookie.
Value: The value of the cookie.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information on the rule displays in the Matching Item tab of the Rule Details dialog box.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

CGI Parameters sub-item

Lists CGI parameters sent along with the request made by the Action item.

Form View Fields

The form-view (bottom pane) for CGI Parameter items lists the following information. Both these fields can be edited and variablized. You can also insert additional CGI Parameter items.

General

Name: The name of the CGI Parameter.
Value: The value of the CGI Parameter.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information on the rule displays in the Matching Item tab of the Rule Details dialog box.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

NTLM Authentication sub-item

Sometimes the pages being requested require NTLM Authentication, that is, the user will be presented with a dialog box asking for a UserID, Password, and Domain. This information is recorded and listed in the tree-view under the Action item that requires it.

Form View Fields

The form-view (bottom pane) for NTLM Authentication items lists the following information:

User: User name typed into the authentication dialog.
Password: Password typed into the authentication dialog.
Domain: Domain typed into the authentication dialog.

Basic Authentication sub-item

Sometimes the pages being requested require Basic Authentication, that is, the user will be presented with a dialog box asking for a UserID and Password. This information is recorded and presented in the tree-view under the Action item which requires it.

Form View Fields

The form-view (bottom pane) for Basic Authentication items lists the following information:

User: User name typed into the authentication dialog.
Password: Password typed into the authentication dialog.

Action item sub-elements

The following items can exist under Action items in the tree-view:

Http Headers sub-item

Cookies sub-item

CGI Parameters sub-item

NTLM Authentication sub-item

Basic Authentication sub-item

Forms

Forms

Many pages that are used during a WWW load testing session contain forms that a user must fill out and submit buttons that get clicked. QALoad identifies forms and the elements within them, as well as determining which submit button was clicked if there is more than one.

When a page contains a form that will be submitted, then a Fill In Form item is inserted into that page's list of items in the tree-view (left pane). Underneath the Fill In Form item are Form Element items, such as Edit Boxes, Radio Buttons, and Check Boxes. Following the Fill In Form item is either a Click On Button item or a PostTo item. For details about Form Elements, see Form sub-items.

When a Fill In Form item is selected in the tree-view (left pane), Visual Navigator highlights the form with a blinking frame in the browser-view (top pane).

Form sub-items

A Fill In Form item in the tree-view can contain a number of sub-items, representing elements that can appear in forms on HTML pages.

The following sub-items can appear under a Fill In Form item:

Hidden sub-item

Editbox sub-item

Selectbox sub-item

TextArea sub-item

Checkbox sub-item

Radio sub-item

Hidden sub-item

Forms can contain hidden fields that do not show up on the page. These fields are not visible to the end user interacting with the browser, but they may need to be variablized for a load test, for example a field that contains a session ID may need to be variablized.

Form View Fields

The form-view (bottom pane) for a Hidden Field element lists the following information:

Name: The name of the hidden field
Value: The value of the hidden field.
Allow this hidden field to be variablized: Select to variablize this field. Click the var... button to select a variable.

Editbox sub-item

One of the more common elements in a form is an edit box. When this tree item is selected, QALoad will draw a blinking frame around the appropriate edit box in the browser-view (top pane). The edit box in the browser-view will show the value that was originally typed in when the transaction was recorded.

Form View Fields

The form-view (bottom pane) for an Edit Box element lists the following information:

General

Name: The name of the edit box.
Value: The value of the edit box. Any changes made to this field will be reflected in the edit box in the browser window.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information in the Matching Item tab of the Rule Details dialog box displays.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

Selectbox sub-item

A select box is often called a drop down selection box or list box. The form-view will appear slightly different depending upon whether the Select Box is capable of supporting multiple selections or not.

Form View Fields

The form-view (bottom pane) for a Select Box element lists the following information:

Name: The name (in the HTML) of the select box.
Items from the Select Box: Lists the items present in the Select Box in the browser-view. An item has a checkbox next to it to indicate if it has been selected. To change a selection, select or clear the checkbox. Your choices will be reflected in the browser. If the Select Box only supports one selection, then only the most recent selection is selected.
Variablized Selections: Edit boxes that allow the use of variables (local or from a datapool) to specify what options are chosen from the Select Box. For a multiple selection Select Box, it is possible to add up to six variables in addition to any item chosen using the check boxes.

For a single selection Select Box, a single edit box is provided to allow you to use a variable (local or from a datapool) to specify the option you want chosen from the Select Box.

TextArea sub-item

A Text Area item is a multi-line text box.

Form View Fields

The form-view (bottom pane) for a TextArea element lists the following information:

General

Name: The name of the Text Area field.
Value: The value of the Text Area field. Any changes you enter into this edit box are reflected in the browser-view (top pane). To enter a linefeed, press `Ctrl+Enter`.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information in the Matching Item tab of the Rule Details dialog box displays.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

Checkbox sub-item

The form-view (bottom pane) for a Checkbox element lists the following information:

Name: The name of the Checkbox.
Value: The value of the Checkbox.

State: Reflects whether the box is checked (selected) or not. If the State is 1 (checked), then the Name and Value are passed along in the request to the server. If the State is 0 (not checked) then the Name and Value are not passed along. You can change the value of the State by clicking on the checkbox control in the browser-view (top pane).

Radio sub-item

Form View Fields

The form-view (bottom pane) for a Radio Button element lists the following information:

General

Group Name: The Group Name is shared by all radio buttons that belong to the same group.
Value: The Value field is what differentiates one radio button from another. The group name and value of the selected radio button will be sent along with the request to the server. The Value of a radio button can be, and often is, different than the text shown in the browser.
use this value button: When you select a radio button in the browser-view (top pane) its value will display in this text box. Click this button to transfer this value into the above Value field.

Matching Parameter Rules

Rule: Lists the rules that have been created for this variable type. These rules may or may not have been placed in the Rule Library.

Applied to Item: Indicates whether the rule is applied to this variable.

GoTo Rule: Goes to the individual rule under the Parameterization Rules tree item. In the right-hand pane, information in the Matching Item tab of the Rule Details dialog box displays.

Previous Match: Goes to the next matching variable immediately preceding the current item in the script.

Next Match: Goes to the next matching variable immediately following the current item in the script.

Apply: Applies the rule highlighted in the listbox to this variable. Only one rule can be applied to an instance of a variable.

Undo Apply: Removes application of the rule highlighted in the listbox from this variable. Other matching variables to which this rule is applied are not affected.

XML Requests

XML document-view

When you click on an XML Request item in the tree-view (left pane) the right pane becomes a document-view displaying a tree-view of details about the XML document requested or returned as the result of an XML request. Each individual XML item appears as a node in the XML document tree. XML elements can have child elements and these appear as child nodes of the XML element. Attributes of an element appear as child nodes of the element, with the attribute value appearing as a child of the attribute name.

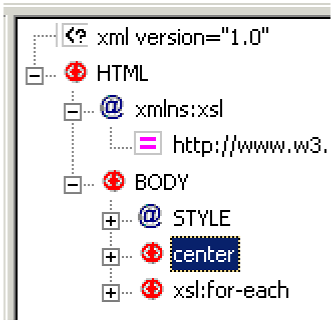
What if no XML data is associated with a request?

If there is no XML document associated with the XML request (for example, an HTTP GET) a message indicating that there is no XML to be displayed is shown in the XML document view.

How does the document-view relate to the form-view?

Selecting an item in the XML document tree will display the form-view details corresponding to that XML element type in the bottom pane.

Following is an example of XML request data displayed in a portion of the XML document-view:



XML form-view

When an XML request is displayed in the document-view (top pane) — as a result of an XML request item or XML reply child item being selected in the Visual Navigator tree-view — you can click on items in the document-view to display information about each in the form-view (bottom pane). If no XML item is selected in the document-view, the XML Page form-view will be displayed instead. For XML items, the form view display options depend on two things:

- what type of XML item is selected in the Visual Navigator tree-view (left pane): an XML request or an XML reply
- what type of XML item is subsequently selected in the XML document-view (top pane).

When an XML item is selected in the XML document-view, the value of that XML item is displayed in an edit box in the form-view. Some values — attribute values and text values — can be edited or variablized (that is, substituting one or more variables for the value in an XML request or selecting the return value from an XML reply item to be received by a variable for later use in the script). Text items, which are values between element tags, and attribute values represent volatile items in an XML document structure, used for passing values to and from Web Services, for example.

The following tables list the possible actions for XML items displayed in a form-view. Valid actions are determined by the XML item type and whether the item is from an HTTP POST request or from an HTTP reply.

In the following tables:

- If an item is *editable*, the value in the form-view can be changed and the new value will be used during replay.
- If a value can be *variablized*, a variable can be substituted for all or part of the value. The variable's value will be placed in the variable's location at replay. An example is a value received from an item from a previous XML document reply.
- If a variable can *receive* a replay value, the return value for the item can be placed into a selected variable during replay. The variable can then be substituted for an input value in a later XML request.

XML Request Items		
XML Request Item	Editable?	Can the Value be Variablized?
Declaration	No	No
DTD (Document Type Definition)	No	No

PI (Processing Instruction)	No	No
Comment	No	No
Element	No	No
Attribute (Name)	No	No
Attribute (Value)	Yes	Yes
Text	Yes	Yes

XML Reply Items	
XML Request Item	Can Variable Receive Replay Value
Declaration	No
DTD (Document Type Definition)	No
PI (Processing Instruction)	No
Comment	No
Element	No
Attribute (Name)	No
Attribute (Value)	Yes
Text	Yes

Using Visual Navigator

Creating a Visual Navigator Script

Creating a Visual Navigator script

Using the Script Development Workbench, you can quickly and easily:

- Convert an old script to a Visual Script How?
- Record a new Visual Script How?

Converting previously-recorded scripts

You can convert existing C-based scripts to a Visual Navigator script.

To convert a previously-recorded script:

1. With a WWW session open, choose **Options>Convert**.
2. On the WWW tab, select the option **Enable Visual Navigator**.
3. Click **OK**.
4. On the Workspace Pane's Capture tab, right-click on the capture file to convert to a script. From the context menu, choose **Convert** (or **Convert As**).
5. You may be prompted to overwrite an existing script file. Click **Yes**.

The Script Development Workbench converts your capture file to a Visual Navigator script, opening it in the Workbook Pane.

What does a Visual Script look like?

Recording a Visual Navigator script

You record a Visual Navigator script the same way you record a regular QALoad script — by setting options to determine the behavior of QALoad while recording, and then clicking through a transaction to mimic a user. QALoad will record all sent and received HTTP and SSL calls using the Script Development Workbench's Web proxy and write the activity to a capture file.

After recording, the capture file must be converted to an editable, C-based script. This is the point where Visual Scripting differs from a regular WWW script. By setting a single option before converting the capture file to an editable script, you can turn your capture file data into a Visual Script that allows you to view the actual Web pages you recorded in a browser-like interface, where you can manipulate the transaction and easily insert variable information into your script without directly editing a line of code.

To record a Visual Script:

1. Open a WWW session in the Script Development Workbench.
2. Click **Options>Record**. The WWW Record Options dialog box opens. Set any relevant options.
3. Click **OK**.
4. For convenience, set conversion options now also. (You can set conversion options after recording your transaction, if you prefer, or even change pre-set options at any time after recording and then re-convert the capture file to apply the changes.):
   a. Click **Options>Convert**. The Universal Convert Options dialog box opens.
   b. Set any applicable options on the General Convert and WWW tabs.
   c. On the WWW tab, select the **Enable Visual Navigator** check box, which will ensure your capture file is converted to a Visual Script later.
   d. Click **OK**.
5. From the toolbar, click the **Start Record** button. *QA*Load launches your application and any proxies, if necessary, and begins recording calls.
6. Record the transaction.
7. When you are finished, click the **Stop Record** button. You will be prompted to save your capture file. By default, capture files (.cap) are saved in the directory `QALoad\Middlewares\WWW\captures`.
8. If you previously set options to prompt automatic conversion, your capture file will be converted to a Visual Script and will open automatically in the editor. For more information about setting up automatic conversion, see Configuring the Script Development Workbench.

   If not, you will be prompted to save and name your file. When you are finished, click **OK**.
9. Click the Script Development Workbench's Captures tab and locate the capture file you just saved.

10. Right-click the file and choose **Convert**. *QA*Load will convert your capture file to a Visual Navigator script and open it in the editor.

Visual Navigator files

When you create a script using Visual Navigator, QALoad saves important information about your script in the following files. These files are saved in the directory `\Compuware\QALoad\Middlewares\WWW` in the subdirectories `\Scripts` and `\Captures`. Some of these files can be modified, and can be opened from the Script Development Workbench's Workspace pane, if necessary.

Filename	Description
Files Generated From Recording	
`<`*`filename`*`>.cap`	A file containing all of the requests and responses that were recorded.
`<`*`filename`*`>.rfd`	Replies to subrequests, which mostly consist of images, style sheets, and javascripts. This data is used to visually recreate the pages as they appeared when recording.
Files Generated From Conversion to a Visual Script	
`<`*`filename`*`>.vistree`	Contains most of the elements of the Visual Navigator tree, including any elements that you modify later or add to your script.
`<`*`filename`*`>.VisHtml`	Contains the HTML pages of all the main requests as well as images, stylesheets, and other subrequested pages. This data is used to visually recreate the pages as they appeared when recording.
`<`*`filename`*`>.VisXml`	Contains any XML/SOAP information that was recorded.
`<`*`filename`*`>.cpp`	A C++ representation of your script.

Inserting Script Items

Inserting tree items

You can insert a number of script items into your converted script.

To insert tree items:

1. Do one of the following:
   - From the Script Development Workbench main menu, choose **Visual Navigator>Insert Tree Item**.

- Right-click in the tree-view (left pane) and choose Insert Tree Item.

2. Choose the item to insert.

Most of the inserted items can be moved up and down the tree using the Up/Down arrows in the form-view (bottom pane) for that item. You can also delete an item highlighted in the tree-view by choosing Delete Tree Item from the menu.

The following script items can be inserted from the Visual Navigator menu:

Extract String

Cookie

Http Header

Content Check

CGI Parameter

Synch

IP Spoof

Read Datapool

Checkpoint pair

Increment Variable/Decrement Variable/Reset Variable

Print Values (debugging)

Comment

Inserting cookies into a script

Cookie items can be added directly to the Html Page item they apply to, under the Action item (for example, a Click on Link item).

To insert a cookie item:

1. In the Visual Navigator tree-view (left pane), navigate to the Html Page item requiring the cookie and then click on it to select it.
2. From the menu, choose **Visual Navigator>Insert Tree Item>Cookie**. A Cookie form-view opens in the bottom pane.
3. In the **Name** field, type a name for the new Cookie or click **Var Wiz...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
4. In the **Value** field, type a value for the new Cookie or click **Var Wiz...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
5. Click **Save** to save your changes.

The Cookie item is added to the script for the selected Html Page item.

Inserting HTTP headers into a Visual Navigator script

You can insert HTTP headers under the Common Http Headers tree item.

To insert a new Http Header item:

1. In the Visual Navigator tree-view (left pane), navigate to the **Common Http Headers** script item, and then click on it.
2. From the menu, choose **Visual Navigator>Insert Tree Item>Http Header**. An **Http Header** form-view opens in the bottom pane.
3. In the **Name** field, type a name for the new header or click **Var Wiz...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
4. In the **Value** field, type a value for the new header or click **Var Wiz...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
5. Click **Save** to save your changes.

The header item is added to the script, and will be used for all requests at playback unless it is overwritten by a header with the same name underneath an individual request action.

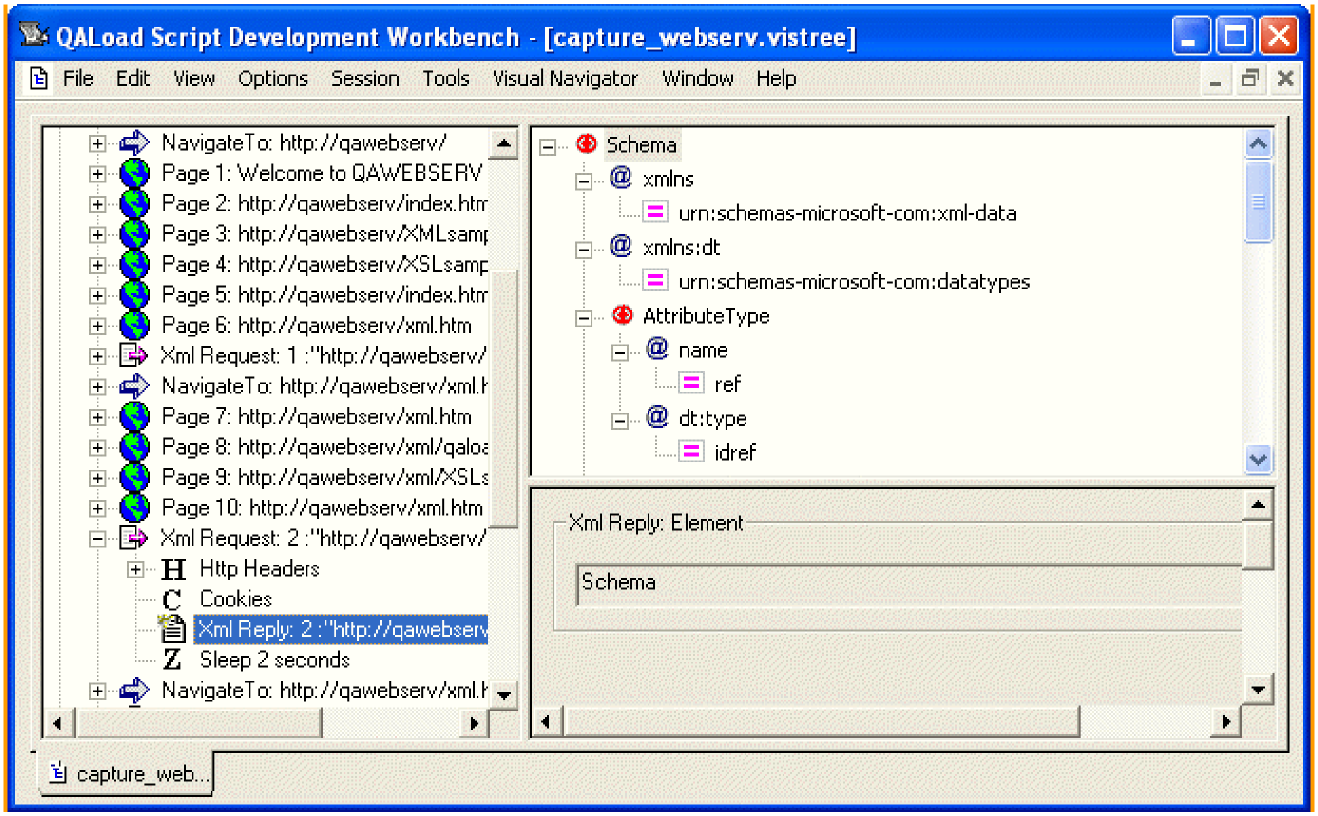
XML Support

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use, visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request or an HTTP POST request with an XML document as the post data, the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

> Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the View menu.



XML requests

When an HTTP request is for an XML document, either in the form of an HTTP GET request, or an HTTP POST request with an XML document as the post data, then an XML Request tree item is displayed. in the tree-view (left pane). The form-view (bottom pane) for an XML Request item includes the following fields:

Reply Status: The reply status code. For most pages which were returned correctly this will be 200 OK.

Request URI: This read-only field shows the URI which was requested that resulted in this page being displayed.

Checkpoint Name: If the page has a title, then it will be used as the checkpoint name. If not, the word Checkpoint will be used. To make sure all checkpoint names are unique, QALoad will add a number to the beginning of the checkpoint name.

XML Request sub-items

An XML Request item can contain the following sub-items.

XML Reply

The URI of the document returned as a result of the XML request. XML data corresponding to the reply is displayed in the browser-view.

HTTP Headers

If a header exists under an action item then it will be sent for that request only. If the header has the same name as one of the common headers, then it will override the common header for this request only. The form-view (bottom pane) for an HTTP header lists its name and value. Because there is no XML data recorded for a header, the browser-view remains empty. It is possible to insert additional HTTP Headers.

Cookies CGI Parameter

The Cookies tree item will contain a list of Cookie items that were sent in the header of the request that this item made while being recorded. Cookies are added automatically by the browser based on the URI that is being requested. They are either set as a result of the previous reply (server returned a Set-Cookie command), or they are set by JavaScript contained in the previous reply. The form-view for a cookie item lists its name and value. Because there is no XML data recorded for a header, the browser-view remains empty.

XML document-view

When you click on an XML Request item in the tree-view (left pane) the right pane becomes a document-view displaying a tree-view of details about the XML document requested or returned as the result of an XML request. Each individual XML item appears as a node in the XML document tree. XML elements can have child elements and these appear as child nodes of the XML element. Attributes of an element appear as child nodes of the element, with the attribute value appearing as a child of the attribute name.

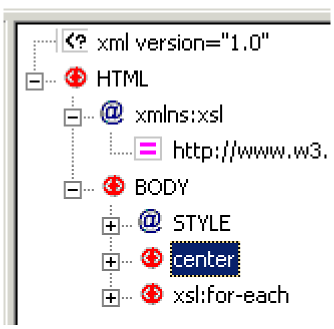
What if no XML data is associated with a request?

If there is no XML document associated with the XML request (for example, an HTTP GET) a message indicating that there is no XML to be displayed is shown in the XML document view.

How does the document-view relate to the form-view?

Selecting an item in the XML document tree will display the form-view details corresponding to that XML element type in the bottom pane.

Following is an example of XML request data displayed in a portion of the XML document-view:



XML form-view

When an XML request is displayed in the document-view (top pane) — as a result of an XML request item or XML reply child item being selected in the Visual Navigator tree-view — you can click on items in the document-view to display information about each in the form-view (bottom pane). If no XML item is selected in the document-view, the XML Page form-view will be displayed instead. For XML items, the form view display options depend on two things:

- what type of XML item is selected in the Visual Navigator tree-view (left pane): an XML request or an XML reply
- what type of XML item is subsequently selected in the XML document-view (top pane).

When an XML item is selected in the XML document-view, the value of that XML item is displayed in an edit box in the form-view. Some values — attribute values and text values — can be edited or variablized (that is, substituting one or more variables for the value in an XML request or selecting the return value from an XML reply item to be received by a variable for later use in the script). Text items, which are values between element tags, and attribute values represent volatile items in an XML document structure, used for passing values to and from Web Services, for example.

The following tables list the possible actions for XML items displayed in a form-view. Valid actions are determined by the XML item type and whether the item is from an HTTP POST request or from an HTTP reply.

In the following tables:

- If an item is *editable*, the value in the form-view can be changed and the new value will be used during replay.
- If a value can be *variablized*, a variable can be substituted for all or part of the value. The variable's value will be placed in the variable's location at replay. An example is a value received from an item from a previous XML document reply.
- If a variable can *receive* a replay value, the return value for the item can be placed into a selected variable during replay. The variable can then be substituted for an input value in a later XML request.

XML Request Items		
XML Request Item	Editable?	Can the Value be Variablized?
Declaration	No	No
DTD (Document Type Definition)	No	No
PI (Processing Instruction)	No	No
Comment	No	No
Element	No	No
Attribute (Name)	No	No
Attribute (Value)	Yes	Yes
Text	Yes	Yes

XML Reply Items	
XML Request Item	Can Variable Receive Replay Value
Declaration	No
DTD (Document Type Definition)	No
PI (Processing Instruction)	No
Comment	No
Element	No

Attribute (Name)	No
Attribute (Value)	Yes
Text	Yes

Visual Navigator's Find and Replace feature

Visual Navigator has an enhanced Find/Replace feature that allows you to find occurrences of strings within the tree-view, allowing you to quickly locate and/or replace text. For example, you could find occurrences of Smith and replace them all with the datapool variable {$ Last Name:User Info $}.

Access Find and Replace from the Edit menu, or by pressing `Ctrl+F`.

Find tab

Find what: Type the string to search for, or click the var... button to select a variable to search for. The appropriate tree item will be selected and the found text will be highlighted in the form-view (bottom pane).

Case Sensitive: Select if Visual Navigator should only find strings with case matching that of the string to locate.

var...: Click to access the Select Variable dialog box, where you can choose a local variable or datapool variable to search for.

Find Next: Click to find the next occurrence of the string.

Cancel: Click to cancel the search and close the dialog box.

Replace tab

Find what: Type the string to search for, or click the var... button to select a variable to search for. The appropriate tree item will be selected and the found text will be highlighted in the form-view (bottom pane).

Replace with: Type the string to replace the search string, or click the var... button to select a variable to replace the search string. The replacement string can be a combination of regular text and one or more variables.

var...: Click to access the Select Variable dialog box, where you can choose a local variable or datapool variable to search for.

Case Sensitive: Select if Visual Navigator should only find strings with case matching that of the string to locate.

Allow read-only fields to be changed: Select to replace text in read-only fields without a confirmation dialog box. If this option is not selected, a confirmation dialog box appears for each occurrence of a string in a read-only field from which you can choose whether to replace the field.

Replace: Click to replace the string in the Find what field with the string in the Replace with field.

Replace All: Click to replace all occurrences of the string in the Find what field with the string in the Replace with field. Matching strings that are found in read-only fields will not be replaced.

Find Next: Click to locate the next occurrence of the search string.

Cancel: Click to cancel the search and close the dialog box.

Parameterization

Overview of Parameterization

Parameterization is the process of substituting certain values in a script with variables. When you modify QALoad scripts before replaying them, the modifications usually are repetitive and consistent. Parameterization provides the means for replacing these values with system-generated variables throughout your scripts. Parameterization is used in Visual Scripting for WWW scripts.

About Parameterization

You can use Parameterization in Visual Scripts to substitute certain variables in your script with values you define. When you modify QALoad scripts before replaying them, the modifications usually are repetitive and consistent. Parameterization provides the means for replacing values with system-generated variables throughout your scripts.

The values for variables are derived from a datapool, an extract string, or a calculated value. A calculated variable generates its values dynamically at runtime based on the formula you define. You can build a calculated variable using datapool values, extract strings, or other alphanumeric and numeric elements.

Methods for Parameterizing a Script

Values in a script that you can parameterize are noted in the form-view (bottom pane) with the var button next to the field. This gives you access to the Datapool and Variables dialog box, where you can define values for variables.

Certain values in the script support the Variable Replacement Wizard. The Variable Replacement Wizard simplifies the process of parameterization by taking you through the necessary steps for defining variables for the fields you want to replace. Fields for which the Variable Replacement Wizard is available are shown in the form-view with Var Wiz button next to the field.

Saving Parameters as Rules

You can create and maintain a table of the variables you define by storing them as rules in the Rule Library. Once stored, the script looks for these rules and replaces the value with the parameters you assign to that rule.

Using Variables with Visual Navigator

Naming variables

When you first create a datapool file, the included variables are automatically assigned the default names Var1, Var2, Var3, and so forth.

QALoad allows you to rename those variables with meaningful names that can even include spaces. This makes it much easier to work with datapools. For example, you could name a datapool variable something logical like City, rather than trying to remember that Var4 in your datapool is the City variable.

Renaming variables

You can quickly and easily rename local or datapool variables from the tree-view. Simply highlight the variable under the Datapool Files or Variables tree-view item, and then change the variable name in the resulting form-view (bottom pane).

You can also edit from the Datapools and Variables dialog box. To access it, right-click anywhere in the tree-view and then choose Datapools and Variables from the shortcut menu. Highlight the variable to rename and click the Rename button.

Datapools and variables

Datapools and variables can be added or modified by several methods. To simply create, delete, or modify datapool files and variables at any time while a script is open in the editor, choose Visual Navigator>Datapools and Variables from the menu to access the Datapools and Variables dialog box.

Alternately, the same dialog box will open automatically whenever you are asked to choose a variable or datapool file while working with the script, allowing you to create the variables you need on-the-fly.

> Data that can be variablized is denoted in the form-view (bottom pane) by the var... button. Clicking the var... button will open the Datapools and Variables dialog box.

Types of Variables

Types of Variables

When values in a script are replaced with variables, the variables typically are derived from extract strings, datapools, or calculated variables.

Extract Strings

Insert an Extract String item when you need to extract information from a reply and store it in a variable to use in future requests or simply for logging the information. For example, if a string is located inside a JavaScript or in a hidden tag that is not visible in the browser, and it might change each time this page is requested, use an Extract String to extract the value. Extract strings search on the text preceding and the text following the string you want to extract.

Datapools

Datapools draw values from a file of acceptable values and use a different set of values each time a parameter is replaced. You can select an existing datapool file or create a new datapool file to add to your script from your datapool directory. Each datapool has a list of variables under it representing columns in the datapool file. When you create a datapool, you specify the number of columns (variables) and rows (values) it contains.

The datapool file you choose is added to your script and is listed under Datapool Files in the Visual Navigator tree-view. You can choose to add a central or local datapool to a script.

> Note: You can have only one central datapool file associated with a script, but can have any number of local datapool files.

Central

Central datapools are Conductor-based. They reside on the same workstation as the QALoad Conductor, and are available to any Player system on the network that requests it from the Conductor. You can apply only one central datapool file to a script.

Local

Local datapools are Player-based. They reside on a Player workstation, and are only available to that Player. You can apply any number of local datapool files to a script.

Calculated Variables

Calculated variables are generated dynamically at runtime and are based on a formula you define. For example, you might want each virtual user to have a unique string, such as Smith1, Smith2, and so forth, or you may want to calculate a new value each time through the transaction loop.

Calculated variables are strings built from one or more elements. These can include datapools, local variables, and text, as well as the following:

Date and Time

Insert the date and time in the format you select.

Random Alphanumeric

Substitutes the value with random alphanumeric characters at runtime. Specify the type and length of characters to use. You can select: letters only, numbers only, or both letters and numbers.

The value can be a fixed length, where you specify the number of characters, or variable length, where you specify the minimum and maximum number of characters.

Random Numeric

Substitutes the value with a random number at runtime. You specify the minimum value, maximum value, the number of decimal places, and the number of leading digits. Numbers generated with fewer leading digits are padded with zeros.

Local Variable

A variable with a static value that you set when you create it.

Virtual User Number

The number used to identify the virtual users during a test. You can include a virtual user number (VU) in the calculation. VUs may be absolute (assigned by the Conductor) or relative (assigned by the Player at runtime).

Absolute

The absolute virtual user number is assigned by the Conductor based on the total number of virtual users on all Players. Each virtual user is assigned a number and no numbers are repeated. Insert an absolute virtual user number when it is necessary to use a completely unique virtual user number in place of a variable.

Relative

The relative virtual user number is the number assigned to the virtual user by its Player. Because a test has multiple Players and each Player assigns virtual user numbers from 0-n, a relative virtual user number is only unique on a single Player.

Local Variable

A local variable is a static value that you can substitute wherever variables can be used. You can quickly add local variables to your script from the Visual Navigator tree-view. Insert Increment Variable, Decrement Variable, and Reset Variable items into the tree-view to manipulate the value of any variables.

About Calculated Variables

Calculated variables generate their values dynamically at runtime based on the formula you define for them. They can generate strings that contain any combination of text, random alphanumeric characters, time and date values, and values from other variables. For example, you might want to calculate a new value dynamically each time through the transaction loop rather than use a datapool file of different values. Another example is when you want each virtual user (VU) to have a unique ID string, such as Smith1, Smith2, and so forth. Using calculated variables serves this purpose. You build a calculated variable by choosing the elements you want to include and inserting them in the calculation.

About Datapool Variables

Datapools are text files containing data records that you can substitute in your script in place of variable data. Each datapool comprises a list of variables representing columns in the datapool file.

Data that can be parameterized is denoted in the form-view (bottom pane) by the var... button. Clicking the var... button opens the Datapools and Variables dialog box. Datapools can be Local (specific to a single Player) or Central (available to all Players).

Note: You can parameterize certain variables using the Variable Replacement Wizard.

Central datapool

A central datapool is a datapool that resides on the same workstation as the QALoad Conductor, and is available to any Player system on the network that requests it from the Conductor. A central datapool is controlled by the Conductor, and you use the Conductor to set any options relating to it. You can apply only one central datapool file to a script.

Local datapool

A local datapool is a datapool that resides on a Player workstation, and is only available to that Player. Because a local datapool resides locally and is only available to the local Player, it does not generate any network traffic. You can apply any number of local datapool files to a script.

About Extract Strings

Insert an Extract String item in your script when you need to replace a system-generated value or reuse a system-created value later in a script. You can extract information from a reply and store it in a variable to use in future requests. For example, if a system-created order number is assigned during a transaction that must be used again later in the transaction, you can use an Extract String to store the value and insert it into the script at the appropriate point..

When you select text to store in an extract string, Visual Navigator uses 10 or more characters on either side of the extracted text to make the search string unique enough to find this copy of the extracted text. You can increase or decrease the size of these strings.

About Local Variables

A local variable has a constant value that you assign when you create the variable. It can be substituted wherever variables can be used. You can quickly add local variables to your script by right-clicking in the Visual Navigator tree-view and selecting Datapools and Variables.

Note: You can insert Increment Variable, Decrement Variable, and Reset Variable items into the tree-view to manipulate the value of any variables.

Adding Variables

Adding a Variable

You can quickly variables to your script from the Visual Navigator tree-view. Types of variables are:

- Local variables
- Datapools
- Calculated variables
- Extract Strings

Note: You can insert Increment Variable, Decrement Variable, and Reset Variable items into the tree-view to manipulate the value of any variables.

Creating a Calculated Variable

The following procedure provides step-by-step instructions for creating and editing a new calculated variable using the Datapools and Variables dialog box.

Note: You can also use the Variable Replacement Wizard to create calculated variables.

To create a calculated variable:

1. Access the **Datapools and Variables** dialog box by doing one of the following:
   - Click Visual Navigator>Datapools and Variables.
   - Right-click in the Visual Navigator tree-view, then select Datapools and Variables.
   - In the Visual Navigator tree-view, click the value you want to replace, and then click var... in the form-view.
2. Click **New Calc Variable**. A new calculated variable name appears in the list box with all the other variables assigned to the script. The new variable is added to the Variables section in the Visual Navigator tree-view (left pane).
3. Type a name for the variable, then click **OK**.
4. In the Visual Navigator tree-view, scroll to the Variables section and select the name of the variable you just created. Details of the calculation variable display in the right-hand pane.
5. Click Edit Calculation. The Edit Calculation Variable dialog box displays.
6. Build the calculation variable by clicking a form in the Templates area.
7. Select or type the associated information in the Details area.
8. Click Insert into Calculation, then click OK to save the changes.

Adding a Variable to a Datapool

You can quickly add datapool variables to your script from the Visual Navigator tree-view.

Note: You can insert Increment Variable, Decrement Variable, and Reset Variable items into the tree-view to manipulate the value of any variables.

To add a variable to a datapool:

1. Right-click anywhere in the Visual Navigator tree-view.
2. From the shortcut menu, select **Datapools and Variables**. The Datapools and Variables dialog box displays.
3. Select a Datapool in the tree-view.
4. Click the **New Variable** button. A variable with the default name *Var#* is added to your list of variables in the datapool you selected. You can rename the new variable or accept the default name.
5. Click **OK**. In the Visual Navigator, the new variable is added to the datapool in the tree-view,and a new column for the variable displays in the form-view.
6. Type a value for the variable in each cell of the column.

Creating, Editing, and Importing Datapools

The following sections provide step-by-step instructions for creating and editing datapools, and for importing a datapool file into a script.

To create a new datapool:

1. With a script open, choose Visual Navigator>Datapools and Variables from the menu. The Datapools and Variables dialog box displays.
2. Click New Datapool File.
3. If this script doesn't have a central (Conductor-based) datapool assigned, the Choose Datapool Type dialog box opens. Specify whether the datapool you are about to create should be central (Conductor-based) or local (Player-based), then click OK. The Create New Datapool File dialog box appears.
4. In the Filename field, type the name for the new datapool.
5. Type the number of rows and columns for the datapool file.
6. Click OK to create the datapool. You are returned to the Datapools and Variables dialog box. The Datapools and Variables dialog box now displays any datapool files assigned to the script, including the one you just created. The new datapool is added to the Datapool Files section in the Visual Navigator tree-view (left pane).
7. Click OK.
8. In the Visual Navigator tree-view, scroll to the Datapool Files section of the script and click the name of the datapool file you just created.
9. In the right-hand pane, type values into the datapool fields to be used at test time.

To import a datapool file:

1. Copy your datapool file, which must be in .csv or .dat format, to the `Datapools` directory of the *QA*Load installation.
2. Insert the datapool into a script as described below.

To edit a datapool file:

If you need to make changes to a datapool file, Compuware recommends that you make a backup copy of the file first. If a backup copy is not available, you may need to check all other scripts that use that datapool file and make appropriate changes.

Caution: Because it is possible for more than one script to use the same datapool file, care should be taken when modifying a datapool file's contents. Doing so for one script may cause errors in other scripts that use the datapool file.

1. With your script open, scroll to the Datapool Files section in the Visual Navigator tree-view (left pane).
2. Select the datapool file you want to edit. A table opens in the right-hand pane displaying the contents of the datapool file.
3. Perform any of the following functions:

   ! Edit a cell's contents: Click in a cell and type over the existing contents to edit the value.

- Insert columns or rows: Right-click on a row or column header and choose Insert Row or Insert Column from the menu. Choose whether to insert the new item before or after the selected item, and type the number of items to insert.
- Delete columns or rows: Right-click on a row or column and choose Delete Row or Delete Column from the menu. (Press SHIFT to select multiple contiguous items or CTRL to select multiple non-contiguous items.)
- Rearrange columns or rows: Select one or more columns or rows by clicking on the headers (press SHIFT first to select multiple items). Drag the rows or columns to a new position. As you drag them, a thin red line indicates where the selection are moved.
- Rename a column header (variable):Click on the column name and then type the new name in the Variable Name field.

4. Click File>Save to save your changes.

Inserting Datapool Variables into a Script

You can quickly add an existing datapool to your script from the Visual Navigator tree-view.

To insert a datapool:

1. Right-click anywhere in the tree-view. From the shortcut menu, choose **Datapools and Variables**.
2. Click **Insert Datapool File**.

Note: Data that can be parameterized is denoted in the form-view (bottom pane) by the var... button or the Var Wiz... button. Clicking the var... button opens the Datapools and Variables dialog box. Clicking the Var Wiz... button opens the Parameterization Wizard.

3. If your script doesn't yet include a central datapool file, the **Choose Datapool Type** dialog box opens, where you can designate your datapool file as a central or local datapool. If your script already includes a central datapool, the **Open** dialog box appears.
4. From the **Open** dialog box, navigate to the datapool file to add to your script. Datapool files are normally located in the directory `\Compuware\QALoad\Datapools`.
5. Select the appropriate file and click **Open**.

The file you selected is added to your script and is listed in your Datapools and Variables dialog box, where you can edit or delete it as necessary. If you expand the view of the file in the Datapools and Variables dialog box, you can see the variables (columns) saved in the file. The names for those columns are stored in the datapool file as a comment line. If they are not named, Visual Navigator assigns them the default name Var#.

Creating Extract Strings

You can extract a value from the web page where it first occurs and place it into a variable using an ExtractString tree item. When you insert the tree item, the HTML source for the page appears in the browser-view so that you can highlight the text to extract. When you highlight the string, some of the text preceding and following the string is selected to identify where the string occurs during playback.

To create an extract string by inserting a tree item:

1. Right-click in the Visual Tree where you want to place the extract string. The HTML source for the page appears in the right-hand pane.

2. In the shortcut menu, click **Insert Tree Item>ExtractString**.
3. In the form_view (bottom pane), type the text you want to extract in the **Search for** field. Select **Match Case** to make your search case-sensitive.
4. Click **Find prev** and **Find next** to select the occurrence of the variable you want to replace.
5. Click **Select variable**. The **Datapool and Variable** dialog box displays.
6. Select the variable in which to store the extract string, and click **OK**.
7. Do one of the following:
   - Click Copy from source. This copies the selected text from the source window into the edit boxes. The text immediately before the selected text appears in the Preceded by field. The text immediately after the selected appears in the Followed by field.

     Note: Use the arrows next to the fields to include more or less of the preceding and following text.
   - Click Enter strings manually and type the text before and after the string in the Preceded by and Followed by fields.
8. In the **Extracted** field, click **Next** to view each instance of the variable in the page. The extracted string is added to the Visual Tree.

Adding a Local Variable

You can quickly add local variables to your script from the Visual Navigator tree-view. A local variable can be substituted wherever variables can be used.

Note: You can insert Increment Variable, Decrement Variable, and Reset Variable items into the tree-view to manipulate the value of any variables.

To add a local variable:

1. Right-click anywhere in the Visual Navigator tree-view.
2. From the shortcut menu, select **Datapools and Variables**. The Datapools and Variables dialog box displays.
3. Select the Variables item in the tree-view.
4. Click New Variable. An unnamed variable is added to your list of local variables.
5. Type a name for the new variable.
6. In the Initial Value field, type the value for the variable.
7. Click OK. The local variable is added to the Visual Navigator tree-view under Variables.

Using the Variable Replacement Wizard

How to Use the Variable Replacement Wizard

The Variable Replacement Wizard guides you through the process of defining variables and substituting them for parameters in the script. Using the wizard, you choose which fields of a parameter to replace with variables, define the variable that replaces the field, and select additional parameters to replace with the variable.

When you use the Variable Replacement Wizard, the variable you define is saved as a rule. When you save the rule in the Rule Library, you can apply the same parameters to future scripts. Subsequent captures use the stored rules during the convert process to identify matching parameters.

The Variable Replacement Wizard is available for the Visual Navigator tree items that have a Var Wiz... button associated with them in the form-view (bottom pane). These include:

! CGI Parameter
! Cookie
! Editbox
! TextArea
! Radio button
! Header

To access the Variable Replacement Wizard:

1. In the Visual Navigator, click the tree item you want to parameterize.
2. Click **Var Wiz...** in the form-view. The **Welcome** dialog box of the Variable Replacement Wizard displays.
3. Click **Next** to start the parameterization process.

Step 1: Choose the Fields to Replace

To choose the fields to replace:

1. Select the fields that in the item should get their values from variables. You can select:
    - Name field
    - Value field
    - Both name and value fields
2. Click the down arrow on the **Item Name** box to display options for how the item you are replacing is identified in the script. This allows the wizard to find and replace similar matching items in the Visual Tree. You can select:
    - is equal to (variable to replace)
    - starts with (variable to replace_XXX)
    - ends with (YYY_variable to replace)
    - contains (YYY_variable to replace_XXX)

    The name and current value of the item you select displays in the details area at the bottom of the dialog box.
3. Review your selections in the **Details** area, and click **Next** to proceed to the next step, where you define the variable.

Step 2: Define the Variable that Replaces the Name or Value Field

To define the variable that replaces the Name or Value field:

Notes:

- The variable you define can replace the entire original value or portions of it.
- When you choose to replace Both name and value fields in the previous step, the current step becomes Step 2.1: Define the variable to replace the Name field. When you complete this step and click Next, the dialog box appears again as Step 2.2: Define the variable to replace the Value field. Complete this step for the Value field and click Next to proceed to Step 3.

1. In the **Variable Name** box, type a name for the variable or accept the default name. The default name is *name#*, where # is the number incremented by one for each new variable.
2. Click the down arrow on the **Frequency** box to display selections for when the original value is replaced with the variable. You can select:
   - Once per test
   - Once per transaction (default)
   - Every usage
3. Do one of the following:
   - Type a value for the variable in the Calculation field.
   - In the Templates area, click the type of value you want to use , then select or type the associated details in the Details area of the dialog box. Click a template below to display the associated detail fields:
     - Date and Time
     - Random Alphanumeric
     - Random Number
     - Datapool Variable
     - Local Variable
     - Virtual User Number
     - Extract String

Note: The text you type or the template you select can represent the entire variable, or you can combine it with other elements.

4. Repeat Step 3 until you include all elements for the calculation.
5. Click **Insert into Calculation**.
6. Click **Next** to proceed to the next step, where you select additional parameters to replace. Click **Back** to edit fields in the preceding dialog box.

Step 3: Select Additional Parameters to Replace

Once you identify the item to replace, the Variable Replacement Wizard lists all occurrences of the same item in the Visual Tree. Select the parameters that you want to replace with this variable definition.

To select additional parameters to replace with this variable definition:

1. Click one or both of the following to select the type of parameters to replace:
   - Match only [variable type] parameters - Select this option to display only matching parameters of the same variable type as the original. For example, if the original parameter you are replacing is a cookie, selecting this option replaces only cookie-type parameters.
   - Match only parameters from the host URL - Select this option to display only matching parameters from the same host URL as the original parameter.
2. In the listbox, select the parameters that should be replaced with the variable definition. Click **Select All** to select all the listed parameters.
3. Click **Next** to proceed to the next step, where you save the variable as a rule.

Step 4: Save the Variable as a Rule

The Variable Replacement Wizard saves the newly defined variable as a rule that you can store in the Rule Library. Storing the rule in the Rule Library means that it is automatically added to the Visual Script when matching parameters are found in future capture files during the convert process.

To save the variable as a rule:

1. In the **Rule name** field, type a descriptive name for the rule or accept the default.
2. Review the details in the **Description** box. This is a summary of the elements you used to define the variable.
3. Select **Add to rule library** to store the rule so it can be used again in future capture files.
4. If you add the rule to the Rule Library, do one of the following:
   - Click the down arrow in the Rule Folder list and select the folder name where the rule will be stored.
   - Click New Folder. The Create New Rule Folder dialog box appears. In the Folder Name field, type the name of the new folder, then click OK.
5. Review the **Matching Items** information.
6. To edit any information, click **Back** until you return to the dialog box with details you want to change.
7. Click **Next** to proceed to the **Summary** dialog box.

Summary

To review and create the variable:

1. In the **Summary** dialog box, review the information you entered for the variable. The information appears in three tabs:
   - Variable Properties - Shows the elements you used to define the variable and its frequency of use. Clicking Details displays the Calculation Details dialog box.
   - Items to Replace - Shows the parameters with the same name in the script that you chose to replace with this variable definition.
   - Rule Properties - Shows details of the variable and rule.
2. Click **Back** to return to the appropriate dialog box to edit information.
3. Click **Finish** to create the variable, apply it as a rule, and use it in the script.

Using the Rule Library

Overview of the Rule Library

You can save variable replacements you define as rules. These are stored in the Rule Library, where you can use them in future recordings and reduce repetitive parameterization tasks.

When subsequent captures are converted, the Visual Navigator scans the Rule Library for matching parameters. For example, if a rule parameterizes the value of a CGI Parameter named SessionID, the Visual Navigator scans through the Visual Tree for any CGI Parameters that match the description defined in the rule. If a match is found, the rule is added to the Visual Tree, where you can apply it to the script.

From the Rule Library, you can create rules, edit rules, and view details of individual rules and the folders in which they are stored.

Elements of the Rule Library Dialog Box

Create or modify parameter rules from the Rule Library dialog box. The dialog box contains the following areas:

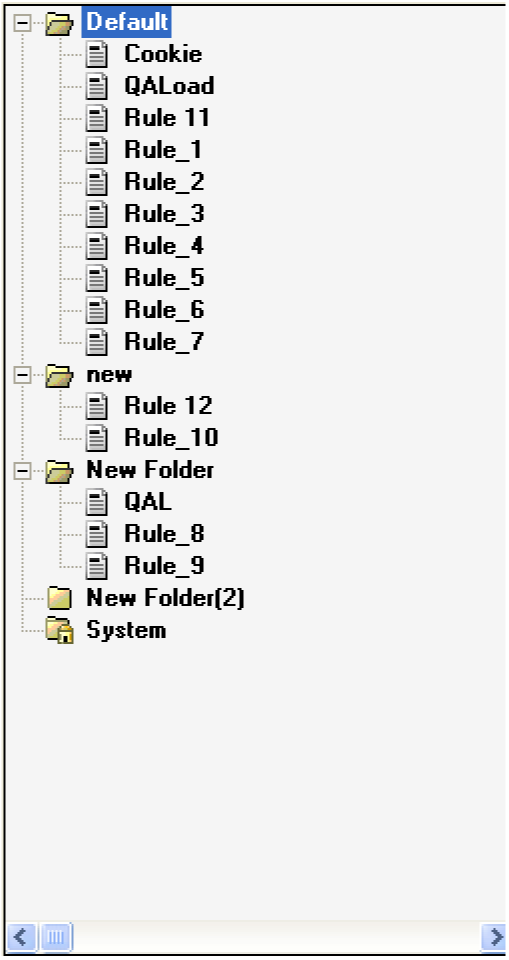
- Menu
- Rule Tree
- Details

Rule Library Menus

Menu Command	Description
File	New Rule - Opens the Rule Creation Wizard. Use the wizard to define a rule that you can use to parameterize future scripts. New Folder - Creates a new folder in the tree-view.
Edit	Use the Edit menu to cut, copy, paste, delete, or rename rules.
Rule	Edit Rule - Opens the Rule Creation Wizard so you can edit a rule you select in the Rule Library.
Help	Displays QALoad help.

Rule Library Tree-View

The Rule Library tree-view displays a hierarchical view of all rules and folders within the Rule Library.



Rule Library Dialog Box Details

The Details area of the Rule Library dialog box displays information on the rules defined in the Rule Library. There are two levels of detail:

- Folder - When you select a folder name in the Rule tree, the **Details** area of the dialog box displays information for each rule in the folder.
- Individual Rule - When you select a rule in a folder in the Rule tree, the **Details** area of the dialog box displays information for the individual rule in three tabs.

Using the Rule Library

Applying a Rule

Follow these steps to apply a rule to items in the script.

To apply a rule from the Visual Tree:

1. In the Visual Navigator vistree, click the **Parameterization Rules** script item and select the rule you want to apply. A description of the rule displays in the form-view.
2. Click the **Matching Items** tab. The properties that determine the matching items and the matching items found in the script display.

3. Do one of the following:
   - Click **Apply Rule** on the left-hand side of the dialog box to apply the rule to all the matching items.
   - Select an item or items in the table and click **Apply to Item** on the right-hand side of the dialog box.

Note: When a different rule has been applied to an individual item, an red exclamation mark (!) appears next to the item in the table. Only one rule can be applied at a time. Click Go to Item and review the information in the Matching Parameter Rules area of the form-view to view the rule applied to the item.

To apply a rule from the form-view for a variable:

1. In the Visual Navigator vistree, select the variable to which you want to apply the rule.
2. In the **Matching Parameter Rules** area at the bottom of the form-view (right-hand pane), select the rule you want to apply.
3. Click **Apply**. The rule is applied to the variable.

Editing a Rule

Follow these steps to edit rules stored in the Rule Library.

To edit a rule:

1. Select a rule in the Rule Library visual tree, and click **Rule>Edit Rule**. The Rule Creation Wizard opens and displays the information on the rule you selected.
2. Follow the procedure for creating a rule to make changes to the rule elements.

Creating Rules from the Rule Library

Create a new rule from the Rule Library using the Rule Creation Wizard.

To open the Rule Library Wizard:

1. Select **Tools>Rule Library**. The **QALoad Rule Library** opens.
2. Select the folder in the tree-view where you want to store the new rule.
3. Click **File>New Rule**. The Welcome dialog box of the **Rule Creation Wizard** displays.

Using the Rule Library Wizard

Step 1 of 3: Choose the Fields to be Replaced

To choose the fields to replace:

1. Choose which fields of the matching item get their values from this rule. You can select:
   - Name field
   - Value field
   - Both name and value fields

2. Click the down arrow on the **Item Name** box to display options for how the item you are replacing is identified in the script. This enables the wizard to find and replace similar matching items. You can select:

- is equal to (variable to replace)
- starts with (variable to replace_XXX)
- ends with (YYY_variable to replace)
- contains (YYY_variable to replace_XXX)

3. Select the parameter type to which you want the rule applied. Choose:

- Only items of the specified type
- All parameter types

4. Select the URLs from which the item should be matched. Choose:

- Match any items of the specified host URL and type the name of the URL.
- Any host

5. Click **Next** to proceed to the next step, where you define the variable.

Step 2 of 3: Define the Variable

To define the variable:

Note: The variable you define can replace the entire original value or portions of it.

1. In the **Variable Name** box, type a name for the variable or accept the default name. The default name depends on the field you chose to replace in the previous step. The default name is:

- NameVar - if you chose to replace the Name field in the previous step.
- ValueVar - if you chose to replace the Value field in the previous step.

Note: When you choose to replace both the Name and Value in the previous step, the default is NameVar. When you complete this step and click Next, the dialog box displays again with ValueVar in the Variable Name field. Complete Step 1 through Step 5 for the Value variable.

2. Click the down arrow on the **Frequency** box to display selections for when the original value is replaced with the rule. You can select:

- Once per test
- Once per transaction (default)
- Every usage

3. In the **Templates** area, click the type of value you want to use , then select or type the associated details in the **Details** area of the dialog box. Click a template below to display the associated detail fields:

- Date and Time
- Random Alphanumeric
- Random Numeric
- Datapool Variable
- Local Variable
- Virtual User Number
- Extract String

Note: The text you type or the template you select can represent the entire variable, or you can combine it with other elements from the templates.

4. Repeat Step 3 and Step 4 until you include all elements for the calculation.

5. Click **Insert into Calculation**.

6. Click **Next** to proceed to the next step, where you select additional parameters to replace. Click **Back** to edit fields in the preceding dialog box.

Note: When you choose to replace Both name and value fields in the previous step, clicking Next repeats the current step so you can define the variable for the Value field. Complete Step 1 through Step 5 for the Value field and click Next to proceed to the next step.

< Previous Next >

Step 3 of 3: Rule Description

Review the rule you created.

To view the rule:

1. Review the details in the **Description** box. This is a summary of the elements you used to define the rule.

   Note: A default name is provided in the Rule Name field. You can type a new, descriptive name for the rule or accept the default.

2. Store the rule in a folder by doing one of the following:
   - Click the down arrow in the Rule Folder list and select the folder name where the rule will be stored.
   - Click New Folder. The Create New Rule Folder dialog box appears. In the Folder Name field, type the name of the new folder, then click OK.
3. Click **Next** to proceed to the **Summary** dialog box.

Summary

To review and create the rule:

1. In the **Summary** dialog box, review the information you entered for the rule. The information is displayed in two tabs:
   - Variable Properties - Shows the elements you used to define the rule and its frequency of use. Clicking Details displays the Variable Details dialog box.
   - Rule Properties - Shows details of the variable and rule.
2. Click **Back** to return to the appropriate dialog box and edit any elements of the rule.
3. Click **Finish** to create the rule and store it in the Rule Library.

EasyScript for WWW

Configuring a Web browser (WWW)

Before you record the WWW requests your Web browser makes, you must configure the browser to use QALoad's proxy server.

To configure a Web browser:

1. Start your Web browser.
2. Specify proxy settings:
   - In the field designated to specify the address of the proxy server, enter the machine name where QALoad Script Development Workbench is installed.
   - In the Port field, enter the port(s) that you specified on the Script Development Workbench's WWW Record Options wizard (Capture Ports fields).
3. Click **OK**.

Streaming media support

QALoad includes support for audio and video download testing of both Windows Media Player and RealOne Player and their supported media formats through a WWW session. When streaming media conversion is enabled and you record a transaction that calls streaming media, an additional command that requests the media is inserted into your script . You do not have to listen to or view the entire media you are requesting. You simply need to record its URL and ensure that the appropriate media player is installed on the Player machines that plays back the script. At run time, the script invokes your media player and requests the streaming media resource. Streaming media through a firewall or proxy server is not supported.

QALoad's streaming media support includes the following media player(s). The appropriate media player must be installed on the machine you are recording from as well as any QALoad Player machine that will be executing the script.

! **RealOne Player** — The media download is initiated by requesting a file that is a data type supported by the RealOne Player. Supported data types are RealAudio, RealVideo, RealText, RealPix, MP3, and SMIL. As a result, the DownloadMediaRP command will be inserted into the script at the appropriate point. At runtime, this command initiates and waits for completion of the download. RealOne Player scripts must be executed as process-based scripts.

! **Windows Media Player** — The media download is initiated by downloading a file with a content type of (audio|video)/(x-ms-asf|s-ms-asf) in the browser. Currently, only .asx files are supported. As a result, the DownloadMediaFromASX command is inserted into the script at the appropriate point.

Note: QALoad does not support scripts that have both RealNetworks media and Windows Media in the same script. To test both types in a single load test, use a different script for each type.

Please note that asynchronous calls may not be played back exactly as they were recorded. For example, if you click on a link in the browser while recording while the media is playing, during replay that link will not be requested until the media clip has finished being processed.

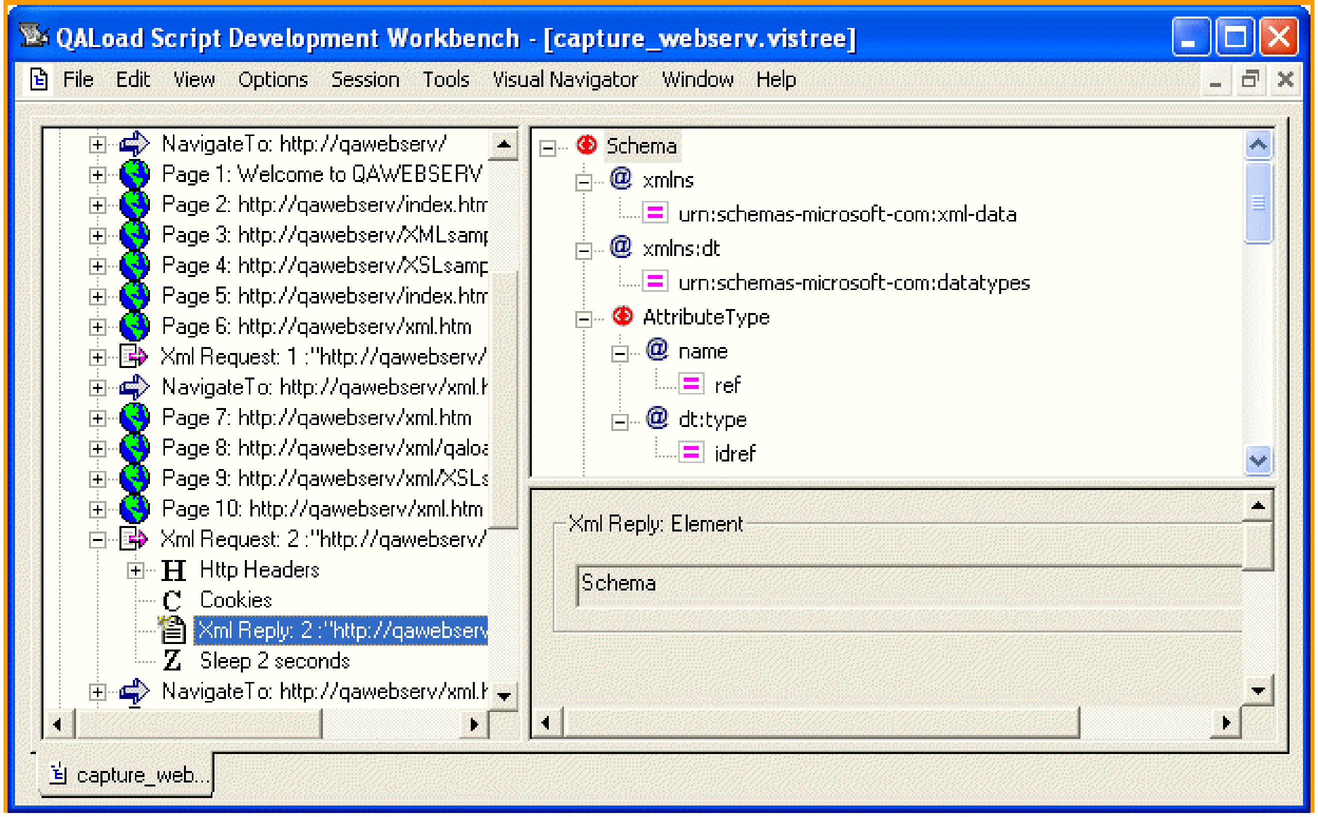
How do I configure QALoad to process streaming media?

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use, visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request or an HTTP POST request with an XML document as the post data, the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the View menu.



CJK Support in QALoad

QALoad supports load testing of Chinese, Japanese, and Korean (CJK) Web applications that use Double Byte Character Sets (DBCS). DBCS is a character set that uses two bytes (16 bits) rather than one byte (8 bits) to represent a single character. Some languages, such as Chinese, Japanese, and Korean, have writing schemes with many different characters and character sets that cannot be represented with single-byte characters such as ASCII and EBCDIC.

QALoad supports the following:

- Simplified Chinese - People's Republic of China (PRC), Singapore
- Traditional Chinese - Taiwan, Hong Kong, Macau
- Japanese
- Korean

Notes:

CJK support only applies to the WWW middleware. Currently, QALoad only supports the CJK Double Byte Character Sets; Web applications that host Bi-Directional (BiDi) characters (which includes Arabic and Hebrew languages) are not currently supported.

UTF-8-encoded characters are treated as an additional character set used on CJK platforms. They are supported when represented on their native operating system. For example, Japanese characters are displayed properly on a Japanese operating system.

QALoad provides two methods of support for CJK: native character and encoding. Depending on your testing requirements and environment, it may be necessary to use both mechanisms to support the load testing of a Web site that contains CJK characters.

- **Native Character:** converts the CJK characters to their original characters in Chinese, Japanese, or Korean. Native character support is only possible when using a native operating system (OS) such as load testing a Japanese Web application from a Japanese version of Microsoft Windows. QALoad supports one CJK language's characters in a script at a time, plus ASCII/English. Native character support is used within test scripts, error messages (generated through system commands that use native characters) and timing files, making script editing easier and timing file analysis quicker.
- **Encoding:** encodes all CJK characters into a sequence of printable characters regardless of the language and exact character set in use. Encoding support is used when load testing multiple language sites from the same OS, or when load testing a CJK site from that of another CJK platform. For example, testing a site with Japanese characters from a Korean or English/ASCII OS. The encoding option is used when native character support cannot be used or when script portability between different CJK language OS is required.

WWW conversion options

Form field as comments: Select this check box to include a comment block in your script that shows each valid field that is encountered after a CGI form has been requested. This option is not available if the Visual Navigator is enabled. Following is an example of a form field comment block:

```
/* Form:1 text Name: name, Value: , Desc: */
/* Form:1 text Name: e-mail, Value: , Desc: */
/* Form:1 text Name: Address, Value: , Desc: */
/* Form:1 text Name: city, Value: , Desc: */
/* Form:1 text Name: state, Value: , Desc: */
/* Form:1 text Name: zip, Value: , Desc: */
/* Form:1 check box Name: echo, Value: , Desc: Echo a copy of HTML page to email */
/* Form:1 radio Name: test, Value: capture, Desc: Capture */
/* Form:1 radio Name: test, Value: replay, Desc: Replay */
/* Form:1 hidden Name: hidden, Value: This rocks!, Desc: */
```

Anchors as comments: Select this check box to include a comment block in your script that shows all the anchors encountered in a requested HTML document. If it is not selected, no anchors will be included in the script. This option is not available if the Visual Navigator is enabled. The following is an example of an anchors comment block:

```
/* Anchors:'http://playback1/standard.html' 'Standard Example' */
/* Anchors:'http://playback1/forms.html' 'Forms Example' */
/* Anchors:'http://playback1/dynamic.html' 'Dynamic HTML Example' */
/* Anchors:'http://playback1/cgi.html' 'CGI Example' */
/* Anchors:'http://playback1/cookies.html' 'Cookies Example’ */
/* Anchors:'http://playback1/ssl.html' 'SSL Example' */
/* Anchors:'http://playback1/javascript.html' 'Java Script Example' */
```

Client Image Maps as Comments: If selected, the command DO_GetClientMapHREF will be inserted into your script followed by a comment block denoting client image maps. This option is not available if the Visual Navigator is enabled.

```
DO_GetClientMapHREF(MAP(1), REGION(1), &ClientMapURL[0]);
```

```
/* Client Map:1 Region:2 HREF: http://www.ethnicgrocer.com/eg/cp/cr.jsp?FOLDER%3C */
/* %3Efolder_id=169991 &ASSORTMENT%3C%3East_id=153827&site=EG&bmUID= */
/* 1005685611375&WebLogicSession=O1GLayxDvTfjLthL65xY6X1cdQIVdCejWCT8wm */
/* D4PLs29z9H7WC wxlkr8f21K1aKoLSMI4Hml6o3|6518173674514870389/ */
/* 167838850/5/80/80/443/443/-1|-8250053020002903791/167838870/5/80 */
/* /80/443/443/-1|6518173674514872679Client Map:1 Region:3 HREF: */
/* http://www.ethnicgrocer.com/eg/cp/cr. */
```

You can search for DO_GetClientMapHREF in the script to help locate the DO_Get and DO_SetValues in the script, as well as where the array of DO_GetClientMapHREF is initialized and freed.

Debug comments: When this option is enabled, comments will be inserted into the converted script to denote replies from the server, anchors, and so on. This option is not available if the Visual Navigator is enabled. For example:

```
/* Received reply:

URL:<http://abcweb.anywhere.com/cafe/default.htm> */
```

Document Title Verification: Periodically, the HTML page title found in a load tested script does not match the HTML page title found in your recorded text. Select this check box to compare the HTML page title contents in your load tested script with the HTML page title contents in your capture file. You can enable the comparison based on prefix- or suffix-specified character match or entire string match.

Entire Document Title: Select this option to compare the entire length of the HTML page title.

Prefix: Select this option to compare the prefix (left-most specified characters) of the HTML page title. Specify the number of characters to match in the Characters to Match field.

Suffix: Select this option to compare the suffix (right-most specified characters) of the HTML page title. Specify the number of characters to match in the Characters to Match field.

Characters to match: After you select the Prefix or Suffix options, use this field to specify the number of characters to match from the HTML page title.

Baud Rate Emulation: Select this check box to download web pages and images at a rate representing the speed of connection, then enter a connection speed in the Baud Rate field. This enables you to simulate modem speed.

Refresh Timeout: Select this check box and type a time value in the field (in seconds) to compare the specified time value against a Web page's META Refresh value (e.g. <META HTTP-EQUIV=Refresh CONTENT="10"; URL="http://www.compuware.com/">). If the META Refresh tag's CONTENT field value is less than the time value you specify, the page is treated as a redirected page. If the CONTENT field value is greater than the time you specify, the page is treated as a regular page.

> Tip: Select this option to avoid infinite loops in the script. Infinite loops can occur if a page refreshes periodically to update data.

Represent CJK as Octal Characters: Select this check box to enable the encoding of captured data from Web applications containing Double Byte Character Sets (DBCS) such as Chinese, Japanese, or Korean. Enabling this option encodes all native characters and is used when native character support cannot be used. Data stays in encoded format throughout the load test: from capture, through convert and replay, to the analysis of the timing file. By leaving this option clear (default), CJK native characters will be used within test scripts, error messages, and timing files, making script editing easier and timing file analysis quicker.

> Note: Native character support can only be used on the same CJK language OS as the application under test. For further information on CJK Support, see CJK Support in QALoad .

Enable Visual Navigator: Select this check box to enable Visual Navigator, producing a visually-oriented script rather than the standard QALoad C-based script.

> Note: When you select this option, some WWW conversion options are not available, such as commenting options, and some automatic playback options.

Advanced button: Accesses the WWW Advanced dialog box, allowing you to set advanced conversion options.

WWW Recording Options

Automatic Startup of Internet Explorer: Select this option to have QALoad automatically launch Internet Explorer, and configure the proxy and port entries for the browser and the QALoad Script Development Workbench. If you do not select this option, you must manually configure your proxy options before recording.

User Started Application: Select this option to configure the application manually, and then set the appropriate options from those that follow.

Proxy Settings for User Started Application

Direct Connection: Select this option if you have a direct connection to the host from which you are recording.

Manual Proxy Configuration: Select this option to specify a proxy through which to connect to the host. Then set the following proxy options:

HTTP: Type the address of the HTTP proxy server. This field has a 255 character limit. Then enter the port number in the Port field. QALoad accepts numbers from 0-65535.

Secure: Type the address of the secure proxy server. This field is only available if you are licensed to use EasyScript for Secure WWW. This field has a 255 character limit. Then enter the port number in the Port field. QALoad accepts numbers from 0-65535.

Exceptions: Type the addresses of any hosts for which QALoad should not use the specified proxies.

Proxy Automatic Configuration Script: Select this option if your proxy should use an automatic configuration script located on your local area network. Then type the script's URL in the URL field.

Advanced Options

Advanced Options: This area displays the settings for several options. To change any of these settings, click the Change Advanced Options button to open the Advanced Options dialog box.

WWW command reference

QALoad provides descriptions and examples of the various commands available for a WWW script. For details, refer to the Language Reference Help section for WWW.

EasyScript for Secure WWW

Overview

EasyScript for Secure WWW supports SSL/HTTPS requests when used in conjunction with the WWW middleware. This support must be purchased separately and is distributed in a separately-installed module.

Importing a client certificate from a Web browser (SSL)

You can import and convert a Client Certificate for any Web site you plan to visit.

To import a client certificate:

1. Start your Web browser.
2. From the browser, select the Client Certificate for the Web site you plan to visit.
3. Export the Client Certificate (.p12 of .pfx file) to a directory where you can access it using the Script Development Workbench.

   Note: When the browser prompts you to enter a password, do not enter a password. If you enter a password, QALoad cannot process the file.
4. Start a WWW Session in the *QA*Load Script Development Workbench.

5. Click **Tools>Maintain Certificates** to open the SSL Certificate Maintenance dialog box.
6. On the **Client Certificates** tab, click the browse button [**...**] to browse for the Client Certificate you want to convert. The Select the Exported Client Certificate to Convert dialog box opens.
7. Make sure **Files of Type** specifies P12 files (*.p12) or PFX files (*.pfx).
8. Select the appropriate Client Certificate and click **Open**. The path and file name of the selected Client Certificate appears in **Enter Certificate to Convert** on the Client Certificates tab.
9. On the **Client Certificates** tab, click **Convert**.
10. Click **Close** to exit the SSL Certificate Maintenance dialog box.

Creating a client certificate in QALoad (SSL)

This procedure assumes you have a WWW session active.

To create a client certificate:

1. From the **Tools** menu, select **Maintain Certificates** to open the **SSL Certificate Maintenance** dialog box.
2. On the Client Certificates tab, enter a name in the **Certificate Name** field.
3. Enter the number of certificates to create.
4. Click the **Create** button to create the *QA*Load Client Certificate. *QA*Load stores it in the `QALoad\Certificates` directory.

Note: On the Unix player platform, you must create the `Certificates` sub-directory in the `QALoad` directory. The directory name is case sensitive.

5. If necessary, configure your Web server to accept *QA*Load as the Certificate Authority. Refer to your Web server documentation for more information.

Creating an SSL Certificate Authority

Note that creating a new CA invalidates all previously created client certificates.

To create an SSL Certificate Authority:

1. Start a WWW session.
2. From the **Tools** menu, select **Maintain Certificates**.
3. Click the Certificate Authority tab.
4. Click the **Create** button to create a new Certificate Authority with the expiration date shown in the field.
5. Exit and re-start the Script Development Workbench.
6. After creating a new Certificate Authority, re-import the CA to your Web server and then create new Client Certificates.

Creating an SSL Server Certificate

To create an SSL Server Certificate:

1. Start a WWW session.
2. From the **Tools** menu, select **Maintain Certificates**.
3. Click the Server Certificate tab.

4. Click the **Create** button to create a new Server Certificate with the expiration date shown in the field.

Sample Scripts

Overview

This section shows examples of how you can manipulate converted scripts to address specific situations or resolve certain problems. Samples include examples of variablization, changes to transaction logic, and detailed descriptions of commonly used commands. Sample scripts are shown for these middlewares:

Citrix

Oracle Forms Server

SAP

Winsock

WWW

Citrix Scripts

Citrix Script Samples

You can address specific situations or resolve certain problems by modifying converted Citrix scripts. The samples shown here include a description of the problem, the procedure for implementing the modification, and samples of a modified script. Script modifications are discussed for:

Handling Citrix Server Farms

Handling Dynamic Window Titles

Handling Intermittent Windows

Handling Unexpected Events

Moving Citrix Connect and Disconnect Outside the Transaction Loop

Scripting Mouse Actions

Using the CtxWaitForScreenUpdate Command

Handling Citrix Server Farms

Handling Citrix Server Farms

Load testing requirements may include connecting to a Citrix server farm, where the load balancing feature supports dynamic redirection to a given server at connection time. This load tests the server farm and Citrix load balancing rather than a single server, which can provide a more realistic load test.

In order to record a script that connects to a farm, you must use an ICA file specified in the Citrix Record Options dialog. Since the ICA file should contain all the necessary connection information, the server field should be left blank when recording.

When converted, the CitrixServer variable has a blank space:

```
/* Declare Variables */
const char *CitrixServer   = " ";
```

```
const char *CitrixUsername = "citrix";
const char *CitrixPassword = "~encr~657E06726F697206";
const char *CitrixDomain   = "qacitrix2";
const int CitrixOutputMode = OUTPUT_MODE_NORMAL;
```

The Citrix client ignores this value and uses the ICA file to dynamically retrieve the server name at playback time.

Conclusion

When you use these techniques to set up a Citrix server farm test script, you allow for dynamic server redirection at playback as part of testing a load balanced Citrix server farm.

Sample Script for Handling Citrix Server Farms

The following sample shows a snippet of a Citrix script that sets up a server farm test script.

Sample Script

```
/* Converted using the following options:
 * General:
 *  Line Split                         : 80 characters
 *  Sleep Seconds                      : 1
 *  Auto Checkpoints                   : No
 * Citrix
 *  General Options                    :
 *   Replay Output Mode                : Normal
 *   Enable Counters                   : No
 *  Timeout Value Options              :
 *    Connect Timeout (s)              : 60
 *    Disconnect Timeout (s)           : 60
 *    Window Creation Timeout (s)      : 30
 *    Ping Timeout (s)                 : 20
 *    Wait Point Timeout (s)           : 30
 *  Input Options                      :
 *   Combine Keyboard Input            : Yes
 *   Combine Mouse Input               : Yes
 *  Window Options                     :
 *   Window Verification               : Yes
 *    Window Max Retries               : 5
 *    Window Wait Retries (ms)         : 5000
 *   Enable Wildcard Title Match       : Yes
 */

#define CITRIX_CLIENT_VERSION "8.100.29670"
#define CITRIX_ICO_VERSION    "2.4"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
     /* Declare Variables */
```

```
///////////////////////////////////////////////////////////
     /// Convert sets the Citrix server name to a single blank
/// space. This is only necessary to verify that the script actually is connecting to a
server farm.
///////////////////////////////////////////////////////////
const char *CitrixServer      = " ";

const char *CitrixUsername    = "test";
const char *CitrixPassword    = "~encr~657E06726F697206";
const char *CitrixDomain      = "test";
const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

/* Citrix Window Information Objects */
CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
CtxWI *CWI_2 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
CtxWI *CWI_3 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
CtxWI *CWI_4 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
CtxWI *CWI_5 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
CtxWI *CWI_6 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
CtxWI *CWI_7 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
CtxWI *CWI_8 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
CtxWI *CWI_9 = new CtxWI(0x2006c, "Calculator", 66, 66, 261, 253);

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("capCtxServerFarm.cpp");

CitrixInit(1);

/* Citrix replay settings */
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, CitrixDomain);
CtxSetICAFile("PRD desktop.ica");
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
```

…rest of script…

Handling Dynamic Window Titles

Modifying the Script to Handle Dynamic Window Titles

Window titles can vary based on the state of the application. For example, Microsoft Word windows may appear with the name of a default document title as part of the window name. Other window titles may contain the user's name, or the day or time. When a window name in the record session varies from the name in a validation or playback session, the script fails with a window title mismatch error.

When this error occurs during validation or playback, you must modify the script to recognize the base pattern of the window title name rather than an exact window title. You can do this during conversion by adding CtxWaitForWindowCreate calls to the script for each named window creation event.

When you modify a script for these occurrences, you must:

- Modify the recording process
- Perform an initial validation

- Identify a Match Pattern
- Modify the script
- Revalidate the script

Note: You may need to configure the workbench and player for validation.

To modify the recording process:

1. Record the Citrix session with the proper options selected.
2. Click Options>Convert, then click the Citrix tab.
3. Ensure that the Enable Wildcard Title Match check box is selected.
4. Click OK to convert the recording.

To perform an initial validation after script record and conversion:

1. Click **Session>Validate Script**.
2. When the validation session generates the window title mismatch error, compare the actual window title name to the expected window title name.

Note: Repeated validations should generate a list of one or more window titles that differ from the expected title.

To perform a Match Pattern identification:

Identify a match pattern that compares all actual window titles to original window titles using the list of window titles from validations.

Note: This regular-expression style match pattern uses the normal character set. Asterisks (*) denote 0 to n wildcard characters. Question marks denote one wildcard character.

To modify the script:

Make the script changes when you identify a workable match pattern for the dynamic window.

1. In the script, ensure that the SetEnableWildcardMatching command exists before the SYNCHRONIZE command.
2. Ensure that the SetEnableWildcardMatching parameter value is set to TRUE.

Note: This statement is inserted when you set the Enable Wildcard Title Match option in Options>Convert before converting the recording.

3. Insert a CtxSetWindowMatchTitle command before the CtxWaitForWindowCreate call for each window creation wait event for the dynamic window. The parameters include the Citrix window identifier and the identified match pattern for the dynamic window.

To revalidate the script:

1. Click **Session>Validate Script**.
2. If the CtxWaitForWindowCreate wait event call fails, identify the actual window title returned.
3. Perform the match pattern identification step and re-create a valid wildcard string to match all expected window values.

Conclusion

Following these techniques, you can modify window creation events to recognize dynamic window titles. The sample scripts illustrate how to determine a base pattern for the window title by sequential validations of the script and how to make relevant changes to the script. The first example uses a substring match to recognize the dynamic window title. The second example uses a wildcard match to accomplish this.

Example One

Script Samples: Example One - Using a Substring Match

In this example, the Microsoft Word application generates a dynamic title. The dynamic name is a concatenation of the default document that Word creates at application startup with the name of the application.

Original Window Title (Record)	"Microsoft Word"
Actual Window Title (Validation)	"document1 - Microsoft Word"
Actual Window Title (Validation)	"document 2 - Microsoft Word"

The script is altered to reflect the fact that the string "Microsoft Word" is always part of the window title. The asterisk (*) wildcard is substituted for the default document name.

"Match Pattern" from window titles	"* - Microsoft Word"

Sample: Original Citrix Script

This is an example of an original Citrix script converted from capture. In this example, the Microsoft Word application generates a dynamic title. The dynamic name is a concatenation of the default document that Word creates at application startup with the name of the application. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  dynamicwindow.cpp
 *
 *  Script Converted on April 18, 2005 at 12:13:47 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                          : 80 characters
 *  Sleep Seconds                       : 1
 *  Auto Checkpoints                    : No
 * Citrix
 *  General Options                     :
 *   Replay Output Mode                 : Normal
 *   Enable Counters                    : No
 *  Timeout Value Options               :
 *    Connect Timeout (s)               : 60
 *    Disconnect Timeout (s)            : 60
```

```
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
   CtxWI *CWI_11 = new CtxWI(0x20068, "", 112, 116, 416, 248);
// Note the initial window title of "Microsoft Word" is set here in the script…
   CtxWI *CWI_12 = new CtxWI(0x2006e, "Microsoft Word", -4, -4, 649, 461);
   CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("dynamicwindow.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
```

```
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

CtxPoint(164, 61);  //1113840775.516

// Window CWI_1 ("Warning !!") created 1113840775.516

CtxWaitForWindowCreate(CWI_1, 1969);

DO_MSLEEP(2046);
CtxPoint(332, 359);  //1113840777.629

DO_MSLEEP(63);
CtxClick(CWI_1, 94, L_BUTTON, NONE); //1113840777.723

// Window CWI_2 ("Log On to Windows") created 1113840777.817

CtxWaitForWindowCreate(CWI_2, 93);

DO_MSLEEP(16);
// Window CWI_1 ("Warning !!") destroyed 1113840777.833

CtxType(CWI_2, "citrix");  //1113840786.615

DO_MSLEEP(8547);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840786.678

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113840788.650

DO_MSLEEP(1672);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840788.713

DO_MSLEEP(359);
CtxType(CWI_2, "q");  //1113840789.683

DO_MSLEEP(438);

DO_MSLEEP(1078);
CtxPoint(231, 325);  //1113840790.309

DO_MSLEEP(78);
CtxMouseDown(CWI_2, L_BUTTON, NONE, 231, 325);  // 1113840790.309

CtxMouseUp(CWI_2, L_BUTTON, NONE, 232, 325); //1113840790.388

CtxPoint(232, 325);  //1113840790.544

// Window CWI_3 ("Please wait...") created 1113840790.544

CtxWaitForWindowCreate(CWI_3, 204);

// Window CWI_2 ("Log On to Windows") destroyed 1113840790.544

// Window CWI_4 ("Citrix License Warning Notice") created 1113840790.591

CtxWaitForWindowCreate(CWI_4, 46);

DO_MSLEEP(16);
```

```
// Window CWI_3 ("Please wait...") destroyed 1113840790.607

// Window CWI_5 ("Citrix License Warning Notice") created 1113840790.654

CtxWaitForWindowCreate(CWI_5, 47);

DO_MSLEEP(16);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113840790.669

// Window CWI_6 ("UsrLogon.Cmd") created 1113840790.716

CtxWaitForWindowCreate(CWI_6, 46);

DO_MSLEEP(16);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113840790.732

// Window CWI_7 ("") created 1113840790.967

DO_MSLEEP(234);
// Window CWI_8 ("ICA Seamless Host Agent") created 1113840791.014

CtxWaitForWindowCreate(CWI_8, 47);

// Window CWI_9 ("Program Manager") created 1113840791.030

CtxWaitForWindowCreate(CWI_9, 16);

CtxPoint(347, 249);  //1113840793.080

// Window CWI_10 ("") created 1113840793.080

DO_MSLEEP(16);

DO_MSLEEP(1203);
CtxPoint(571, 256);  //1113840796.133

DO_MSLEEP(1844);
// Window CWI_10 ("") destroyed 1113840796.133

DO_MSLEEP(1828);
CtxPoint(446, 208);  //1113840798.309

DO_MSLEEP(344);
CtxClick(CWI_5, 78, L_BUTTON, NONE); //1113840798.387

DO_MSLEEP(437);
CtxPoint(311, 303);  //1113840798.935

DO_MSLEEP(110);
CtxClick(CWI_5, 109, L_BUTTON, NONE); //1113840799.045

DO_MSLEEP(94);
// Window CWI_5 ("Citrix License Warning Notice") destroyed 1113840799.138

DO_MSLEEP(390);
CtxPoint(275, 1);  //1113840799.718

DO_MSLEEP(188);
CtxClick(CWI_8, 62, L_BUTTON, NONE); //1113840799.780

DO_MSLEEP(703);
CtxPoint(200, 187);  //1113840800.579

DO_MSLEEP(94);
CtxClick(CWI_8, 78, L_BUTTON, NONE); //1113840800.657

DO_MSLEEP(110);
CtxPoint(209, 192);  //1113840800.766
```

```
// Window CWI_8 ("ICA Seamless Host Agent") destroyed 1113840800.766

DO_MSLEEP(5328);
CtxPoint(115, 175);  //1113840806.245

DO_MSLEEP(140);
CtxDoubleClick(CWI_9); // 1113840806.621

CtxMouseUp(CWI_7, L_BUTTON, NONE, 114, 174); //1113840806.512

DO_MSLEEP(16);
CtxPoint(114, 174);  //1113840806.621

DO_MSLEEP(359);
CtxPoint(284, 205);  //1113840807.028

// Window CWI_11 ("") created 1113840807.028

DO_MSLEEP(16);
CtxPoint(309, 208);  //1113840807.326

// Window CWI_12 ("Microsoft Word") created 1113840807.326

/// The window create wait event.  If the window title at replay
/// is any different that "Microsoft Word", this wait event will FAIL.
CtxWaitForWindowCreate(CWI_12, 266);

DO_MSLEEP(46);
CWI_12->setTitle("Document1 - Microsoft Word"); //1113840807.388

DO_MSLEEP(16);
// Window CWI_11 ("") destroyed 1113840807.388

DO_MSLEEP(4188);
CtxPoint(626, 8);  //1113840811.693

DO_MSLEEP(109);
CtxClick(CWI_12, 94, L_BUTTON, NONE); //1113840811.787

DO_MSLEEP(218);
CtxPoint(623, 40);  //1113840812.006

// Window CWI_12 ("Document1 - Microsoft Word") destroyed 1113840812.006

DO_MSLEEP(2204);
CtxPoint(385, 1);  //1113840815.262

DO_MSLEEP(1046);
// Window CWI_9 ("Program Manager") destroyed 1113840815.262

// Window CWI_7 ("") destroyed 1113840815.262

// Window CWI_13 ("Please wait...") created 1113840815.513

CtxWaitForWindowCreate(CWI_13, 0);

DO_SetTransactionCleanup();

CtxDisconnect();

END_TRANSACTION();

delete CWI_1; // "Warning !!"
delete CWI_2; // "Log On to Windows"
delete CWI_3; // "Please wait..."
delete CWI_4; // "Citrix License Warning Notice"
delete CWI_5; // "Citrix License Warning Notice"
delete CWI_6; // "UsrLogon.Cmd"
```

```
    delete CWI_7; // ""
    delete CWI_8; // "ICA Seamless Host Agent"
    delete CWI_9; // "Program Manager"
    delete CWI_10; // ""
    delete CWI_11; // ""
    delete CWI_12; // "Microsoft Word"
    delete CWI_13; // "Please wait..."

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Sample One: Modified Citrix Script

In this example, the Microsoft Word application generates dynamic title that is a concatenation of the default document that Word creates at application startup with the name of the application. This script is altered to reflect the fact that the string “Microsoft Word” is always part of the window title. The asterisk (*) wildcard is substituted for the default document name. Changes to the original script are highlighted in bold.

Sample Script

```
/*
 *  dynamicwindow.cpp
 *
 *  Script Converted on April 18, 2005 at 12:13:47 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : No
 * Citrix
 *  General Options                 :
 *   Replay Output Mode             : Normal
 *   Enable Counters                : No
 *  Timeout Value Options           :
 *    Connect Timeout (s)           : 60
 *    Disconnect Timeout (s)        : 60
 *    Window Creation Timeout (s)   : 30
 *    Ping Timeout (s)              : 20
 *    Wait Point Timeout (s)        : 30
 *  Input Options                   :
 *   Combine Keyboard Input         : Yes
 *   Combine Mouse Input            : Yes
 *  Window Options                  :
 *   Window Verification            : Yes
 *    Window Max Retries            : 5
 *    Window Wait Retries (ms)      : 5000
```

```
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
   CtxWI *CWI_11 = new CtxWI(0x20068, "", 112, 116, 416, 248);
// The initial window title of "Microsoft Word" does not change
   CtxWI *CWI_12 = new CtxWI(0x2006e, "Microsoft Word", -4, -4, 649, 461);
   CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("dynamicwindow.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
   CtxSetEnableCounters(FALSE);
   CtxSetWindowRetries(5, 5000);
   CtxSetEnableWildcardMatching(TRUE);

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   CtxConnect(CitrixServer, CitrixOutputMode);
```

```
CtxPoint(164, 61);  //1113840775.516

// Window CWI_1 ("Warning !!") created 1113840775.516

CtxWaitForWindowCreate(CWI_1, 1969);

DO_MSLEEP(2046);
CtxPoint(332, 359);  //1113840777.629

DO_MSLEEP(63);
CtxClick(CWI_1, 94, L_BUTTON, NONE); //1113840777.723

// Window CWI_2 ("Log On to Windows") created 1113840777.817

CtxWaitForWindowCreate(CWI_2, 93);

DO_MSLEEP(16);
// Window CWI_1 ("Warning !!") destroyed 1113840777.833

CtxType(CWI_2, "citrix");  //1113840786.615

DO_MSLEEP(8547);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840786.678

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113840788.650

DO_MSLEEP(1672);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840788.713

DO_MSLEEP(359);
CtxType(CWI_2, "q");  //1113840789.683

DO_MSLEEP(438);

DO_MSLEEP(1078);
CtxPoint(231, 325);  //1113840790.309

DO_MSLEEP(78);
CtxMouseDown(CWI_2, L_BUTTON, NONE, 231, 325);  // 1113840790.309

CtxMouseUp(CWI_2, L_BUTTON, NONE, 232, 325); //1113840790.388

CtxPoint(232, 325);  //1113840790.544

// Window CWI_3 ("Please wait...") created 1113840790.544

CtxWaitForWindowCreate(CWI_3, 204);

// Window CWI_2 ("Log On to Windows") destroyed 1113840790.544

// Window CWI_4 ("Citrix License Warning Notice") created 1113840790.591

CtxWaitForWindowCreate(CWI_4, 46);

DO_MSLEEP(16);
// Window CWI_3 ("Please wait...") destroyed 1113840790.607

// Window CWI_5 ("Citrix License Warning Notice") created 1113840790.654

CtxWaitForWindowCreate(CWI_5, 47);

DO_MSLEEP(16);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113840790.669

// Window CWI_6 ("UsrLogon.Cmd") created 1113840790.716

CtxWaitForWindowCreate(CWI_6, 46);
```

```
DO_MSLEEP(16);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113840790.732

// Window CWI_7 ("") created 1113840790.967

DO_MSLEEP(234);
// Window CWI_8 ("ICA Seamless Host Agent") created 1113840791.014

CtxWaitForWindowCreate(CWI_8, 47);

// Window CWI_9 ("Program Manager") created 1113840791.030

CtxWaitForWindowCreate(CWI_9, 16);

CtxPoint(347, 249);  //1113840793.080

// Window CWI_10 ("") created 1113840793.080

DO_MSLEEP(16);

DO_MSLEEP(1203);
CtxPoint(571, 256);  //1113840796.133

DO_MSLEEP(1844);
// Window CWI_10 ("") destroyed 1113840796.133

DO_MSLEEP(1828);
CtxPoint(446, 208);  //1113840798.309

DO_MSLEEP(344);
CtxClick(CWI_5, 78, L_BUTTON, NONE); //1113840798.387

DO_MSLEEP(437);
CtxPoint(311, 303);  //1113840798.935

DO_MSLEEP(110);
CtxClick(CWI_5, 109, L_BUTTON, NONE); //1113840799.045

DO_MSLEEP(94);
// Window CWI_5 ("Citrix License Warning Notice") destroyed 1113840799.138

DO_MSLEEP(390);
CtxPoint(275, 1);  //1113840799.718

DO_MSLEEP(188);
CtxClick(CWI_8, 62, L_BUTTON, NONE); //1113840799.780

DO_MSLEEP(703);
CtxPoint(200, 187);  //1113840800.579

DO_MSLEEP(94);
CtxClick(CWI_8, 78, L_BUTTON, NONE); //1113840800.657

DO_MSLEEP(110);
CtxPoint(209, 192);  //1113840800.766

// Window CWI_8 ("ICA Seamless Host Agent") destroyed 1113840800.766

DO_MSLEEP(5328);
CtxPoint(115, 175);  //1113840806.245

DO_MSLEEP(140);
CtxDoubleClick(CWI_9); // 1113840806.621

CtxMouseUp(CWI_7, L_BUTTON, NONE, 114, 174); //1113840806.512

DO_MSLEEP(16);
CtxPoint(114, 174);  //1113840806.621
```

```
    DO_MSLEEP(359);
    CtxPoint(284, 205);  //1113840807.028

    // Window CWI_11 ("") created 1113840807.028

    DO_MSLEEP(16);
    CtxPoint(309, 208);  //1113840807.326

    // Window CWI_12 ("Microsoft Word") created 1113840807.326

//////////////////////////////////////////////////////////////////////
/// The CtxSetWindowMatchTitle command is inserted prior to the
/// CtxSetWindowMatchTitle call. Note that the window identifier is the
//  same for both calls, and the "match pattern" for the command is from
/// Example One in the Match Pattern Identification section above.
//////////////////////////////////////////////////////////////////////

    CtxSetWindowMatchTitle(CW1_12, "*Microsoft Word"" );

/// The window create wait event call does not change.
    CtxWaitForWindowCreate(CWI_12, 266);

    DO_MSLEEP(46);
    CWI_12->setTitle("Document1 - Microsoft Word"); //1113840807.388

    DO_MSLEEP(16);
    // Window CWI_11 ("") destroyed 1113840807.388

    DO_MSLEEP(4188);
    CtxPoint(626, 8);  //1113840811.693

    DO_MSLEEP(109);
    CtxClick(CWI_12, 94, L_BUTTON, NONE); //1113840811.787

    DO_MSLEEP(218);
    CtxPoint(623, 40);  //1113840812.006

    // Window CWI_12 ("Document1 - Microsoft Word") destroyed 1113840812.006

    DO_MSLEEP(2204);
    CtxPoint(385, 1);  //1113840815.262

    DO_MSLEEP(1046);
    // Window CWI_9 ("Program Manager") destroyed 1113840815.262

    // Window CWI_7 ("") destroyed 1113840815.262

    // Window CWI_13 ("Please wait...") created 1113840815.513

    CtxWaitForWindowCreate(CWI_13, 0);

    DO_SetTransactionCleanup();

    CtxDisconnect();

    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "Citrix License Warning Notice"
    delete CWI_6; // "UsrLogon.Cmd"
    delete CWI_7; // ""
    delete CWI_8; // "ICA Seamless Host Agent"
    delete CWI_9; // "Program Manager"
    delete CWI_10; // ""
```

```
    delete CWI_11; // ""
    delete CWI_12; // "Microsoft Word"
    delete CWI_13; // "Please wait..."

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();

}
```

Example Two

Script Samples: Example Two - Using a Wildcard Match

In this example, the Sample Application generates a dynamic title. The dynamic name is the name of the application followed by the time the script is created.

Original Window Title (Record)	"Sample Application – 09:01:23 AM"
Actual Window Title (Validation)	"Sample Application – 11:00:04 AM"
Actual Window Title (Validation)	"Sample Application – 12:20:52 PM"

The question mark (?) wildcard is substituted for a given time.

"Match Pattern" from window titles	"Sample Application – ??:??:?? ?M"

Sample Two: Original Citrix Script

The following sample is an original Citrix script converted from capture. In this example, the Sample Application generates a dynamic title. The dynamic name is the name of the application followed by the time the script is created. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  dynamicwindow2.cpp
 *
 *  Script Converted on April 18, 2005 at 12:13:47 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                        : 80 characters
 *  Sleep Seconds                     : 1
 *  Auto Checkpoints                  : No
```

```
 * Citrix
 *  General Options                  :
 *   Replay Output Mode              : Normal
 *   Enable Counters                 : No
 *  Timeout Value Options            :
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
   CtxWI *CWI_11 = new CtxWI(0x20068, "", 112, 116, 416, 248);
// Note the initial window title of "Sample Application 09:01:23 AM" is set here…
   CtxWI *CWI_12 = new CtxWI(0x2006e, "Sample Application 09:01:23 AM ", -4, -4, 649, 461);
   CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("dynamicwindow.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
```

```
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

CtxPoint(164, 61);  //1113840775.516

// Window CWI_1 ("Warning !!") created 1113840775.516

CtxWaitForWindowCreate(CWI_1, 1969);

DO_MSLEEP(2046);
CtxPoint(332, 359);  //1113840777.629

DO_MSLEEP(63);
CtxClick(CWI_1, 94, L_BUTTON, NONE); //1113840777.723

// Window CWI_2 ("Log On to Windows") created 1113840777.817

CtxWaitForWindowCreate(CWI_2, 93);

DO_MSLEEP(16);
// Window CWI_1 ("Warning !!") destroyed 1113840777.833

CtxType(CWI_2, "citrix");  //1113840786.615

DO_MSLEEP(8547);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840786.678

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113840788.650

DO_MSLEEP(1672);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840788.713

DO_MSLEEP(359);
CtxType(CWI_2, "q");  //1113840789.683

DO_MSLEEP(438);

DO_MSLEEP(1078);
CtxPoint(231, 325);  //1113840790.309

DO_MSLEEP(78);
CtxMouseDown(CWI_2, L_BUTTON, NONE, 231, 325);  // 1113840790.309

CtxMouseUp(CWI_2, L_BUTTON, NONE, 232, 325); //1113840790.388

CtxPoint(232, 325);  //1113840790.544

// Window CWI_3 ("Please wait...") created 1113840790.544
```

```
CtxWaitForWindowCreate(CWI_3, 204);

// Window CWI_2 ("Log On to Windows") destroyed 1113840790.544

// Window CWI_4 ("Citrix License Warning Notice") created 1113840790.591

CtxWaitForWindowCreate(CWI_4, 46);

DO_MSLEEP(16);
// Window CWI_3 ("Please wait...") destroyed 1113840790.607

// Window CWI_5 ("Citrix License Warning Notice") created 1113840790.654

CtxWaitForWindowCreate(CWI_5, 47);

DO_MSLEEP(16);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113840790.669

// Window CWI_6 ("UsrLogon.Cmd") created 1113840790.716

CtxWaitForWindowCreate(CWI_6, 46);

DO_MSLEEP(16);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113840790.732

// Window CWI_7 ("") created 1113840790.967

DO_MSLEEP(234);
// Window CWI_8 ("ICA Seamless Host Agent") created 1113840791.014

CtxWaitForWindowCreate(CWI_8, 47);

// Window CWI_9 ("Program Manager") created 1113840791.030

CtxWaitForWindowCreate(CWI_9, 16);

CtxPoint(347, 249);  //1113840793.080

// Window CWI_10 ("") created 1113840793.080

DO_MSLEEP(16);

DO_MSLEEP(1203);
CtxPoint(571, 256);  //1113840796.133

DO_MSLEEP(1844);
// Window CWI_10 ("") destroyed 1113840796.133

DO_MSLEEP(1828);
CtxPoint(446, 208);  //1113840798.309

DO_MSLEEP(344);
CtxClick(CWI_5, 78, L_BUTTON, NONE); //1113840798.387

DO_MSLEEP(437);
CtxPoint(311, 303);  //1113840798.935

DO_MSLEEP(110);
CtxClick(CWI_5, 109, L_BUTTON, NONE); //1113840799.045

DO_MSLEEP(94);
// Window CWI_5 ("Citrix License Warning Notice") destroyed 1113840799.138

DO_MSLEEP(390);
CtxPoint(275, 1);  //1113840799.718
```

```
    DO_MSLEEP(188);
    CtxClick(CWI_8, 62, L_BUTTON, NONE); //1113840799.780

    DO_MSLEEP(703);
    CtxPoint(200, 187);  //1113840800.579

    DO_MSLEEP(94);
    CtxClick(CWI_8, 78, L_BUTTON, NONE); //1113840800.657

    DO_MSLEEP(110);
    CtxPoint(209, 192);  //1113840800.766

    // Window CWI_8 ("ICA Seamless Host Agent") destroyed 1113840800.766

    DO_MSLEEP(5328);
    CtxPoint(115, 175);  //1113840806.245

    DO_MSLEEP(140);
    CtxDoubleClick(CWI_9); // 1113840806.621

    CtxMouseUp(CWI_7, L_BUTTON, NONE, 114, 174); //1113840806.512

    DO_MSLEEP(16);
    CtxPoint(114, 174);  //1113840806.621

    DO_MSLEEP(359);

    CtxPoint(284, 205);  //1113840807.028

    // Window CWI_11 ("") created 1113840807.028

    DO_MSLEEP(16);
    CtxPoint(309, 208);  //1113840807.326

    // Window CWI_12 ("Microsoft Word") created 1113840807.326

/// The window create wait event.  If the window title at replay
/// is any different that "Sample Application 09:01:23 AM", this wait event
/// will FAIL.
    CtxWaitForWindowCreate(CWI_12, 266);

    DO_MSLEEP(46);
    CWI_12->setTitle("Document1 - Microsoft Word"); //1113840807.388

    DO_MSLEEP(16);
    // Window CWI_11 ("") destroyed 1113840807.388

    DO_MSLEEP(4188);
    CtxPoint(626, 8);  //1113840811.693

    DO_MSLEEP(109);
    CtxClick(CWI_12, 94, L_BUTTON, NONE); //1113840811.787

    DO_MSLEEP(218);
    CtxPoint(623, 40);  //1113840812.006

    // Window CWI_12 ("Document1 - Microsoft Word") destroyed 1113840812.006

    DO_MSLEEP(2204);
    CtxPoint(385, 1);  //1113840815.262

    DO_MSLEEP(1046);
    // Window CWI_9 ("Program Manager") destroyed 1113840815.262
```

```
    // Window CWI_7 ("") destroyed 1113840815.262

    // Window CWI_13 ("Please wait...") created 1113840815.513

    CtxWaitForWindowCreate(CWI_13, 0);

    DO_SetTransactionCleanup();

    CtxDisconnect();

    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "Citrix License Warning Notice"
    delete CWI_6; // "UsrLogon.Cmd"
    delete CWI_7; // ""
    delete CWI_8; // "ICA Seamless Host Agent"
    delete CWI_9; // "Program Manager"
    delete CWI_10; // ""
    delete CWI_11; // ""
    delete CWI_12; // "Microsoft Word"
    delete CWI_13; // "Please wait..."

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Sample Two: Modified Citrix Script

The following is an example of a Citrix script modified to handle a dynamic window title. In this example, the Sample Application generates a dynamic title. The dynamic name is the name of the application followed by the time the script is created. The question mark (?) wildcard is substituted for a given time. Changes to the original script are highlighted in bold.

Sample Script

```
/*
 *  dynamicwindow2.cpp
 *
 *  Script Converted on April 18, 2005 at 12:13:47 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
```

```
 * Citrix
 *  General Options                  :
 *   Replay Output Mode              : Normal
 *   Enable Counters                 : No
 *  Timeout Value Options            :
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
   CtxWI *CWI_11 = new CtxWI(0x20068, "", 112, 116, 416, 248);
// The initial window title of "Sample Application 09:01:23 AM" does not change
   CtxWI *CWI_12 = new CtxWI(0x2006e, "Sample Application 09:01:23 AM", -4, -4, 649, 461);
   CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("dynamicwindow.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
```

```
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

CtxPoint(164, 61);  //1113840775.516

// Window CWI_1 ("Warning !!") created 1113840775.516

CtxWaitForWindowCreate(CWI_1, 1969);

DO_MSLEEP(2046);
CtxPoint(332, 359);  //1113840777.629

DO_MSLEEP(63);
CtxClick(CWI_1, 94, L_BUTTON, NONE); //1113840777.723

// Window CWI_2 ("Log On to Windows") created 1113840777.817

CtxWaitForWindowCreate(CWI_2, 93);

DO_MSLEEP(16);
// Window CWI_1 ("Warning !!") destroyed 1113840777.833

CtxType(CWI_2, "citrix");  //1113840786.615

DO_MSLEEP(8547);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840786.678

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113840788.650

DO_MSLEEP(1672);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113840788.713

DO_MSLEEP(359);
CtxType(CWI_2, "q");  //1113840789.683

DO_MSLEEP(438);

DO_MSLEEP(1078);
CtxPoint(231, 325);  //1113840790.309

DO_MSLEEP(78);
CtxMouseDown(CWI_2, L_BUTTON, NONE, 231, 325);  // 1113840790.309

CtxMouseUp(CWI_2, L_BUTTON, NONE, 232, 325); //1113840790.388

CtxPoint(232, 325);  //1113840790.544

// Window CWI_3 ("Please wait...") created 1113840790.544
```

```
CtxWaitForWindowCreate(CWI_3, 204);

// Window CWI_2 ("Log On to Windows") destroyed 1113840790.544

// Window CWI_4 ("Citrix License Warning Notice") created 1113840790.591

CtxWaitForWindowCreate(CWI_4, 46);

DO_MSLEEP(16);
// Window CWI_3 ("Please wait...") destroyed 1113840790.607

// Window CWI_5 ("Citrix License Warning Notice") created 1113840790.654

CtxWaitForWindowCreate(CWI_5, 47);

DO_MSLEEP(16);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113840790.669

// Window CWI_6 ("UsrLogon.Cmd") created 1113840790.716

CtxWaitForWindowCreate(CWI_6, 46);

DO_MSLEEP(16);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113840790.732

// Window CWI_7 ("") created 1113840790.967

DO_MSLEEP(234);
// Window CWI_8 ("ICA Seamless Host Agent") created 1113840791.014

CtxWaitForWindowCreate(CWI_8, 47);

// Window CWI_9 ("Program Manager") created 1113840791.030

CtxWaitForWindowCreate(CWI_9, 16);

CtxPoint(347, 249);  //1113840793.080

// Window CWI_10 ("") created 1113840793.080

DO_MSLEEP(16);

DO_MSLEEP(1203);
CtxPoint(571, 256);  //1113840796.133

DO_MSLEEP(1844);
// Window CWI_10 ("") destroyed 1113840796.133

DO_MSLEEP(1828);
CtxPoint(446, 208);  //1113840798.309

DO_MSLEEP(344);
CtxClick(CWI_5, 78, L_BUTTON, NONE); //1113840798.387

DO_MSLEEP(437);
CtxPoint(311, 303);  //1113840798.935

DO_MSLEEP(110);
CtxClick(CWI_5, 109, L_BUTTON, NONE); //1113840799.045

DO_MSLEEP(94);
// Window CWI_5 ("Citrix License Warning Notice") destroyed 1113840799.138

DO_MSLEEP(390);
CtxPoint(275, 1);  //1113840799.718
```

```
    DO_MSLEEP(188);
    CtxClick(CWI_8, 62, L_BUTTON, NONE); //1113840799.780

    DO_MSLEEP(703);
    CtxPoint(200, 187);  //1113840800.579

    DO_MSLEEP(94);
    CtxClick(CWI_8, 78, L_BUTTON, NONE); //1113840800.657

    DO_MSLEEP(110);
    CtxPoint(209, 192);  //1113840800.766

    // Window CWI_8 ("ICA Seamless Host Agent") destroyed 1113840800.766

    DO_MSLEEP(5328);
    CtxPoint(115, 175);  //1113840806.245

    DO_MSLEEP(140);
    CtxDoubleClick(CWI_9); // 1113840806.621

    CtxMouseUp(CWI_7, L_BUTTON, NONE, 114, 174); //1113840806.512

    DO_MSLEEP(16);
    CtxPoint(114, 174);  //1113840806.621

    DO_MSLEEP(359);

    CtxPoint(284, 205);  //1113840807.028

    // Window CWI_11 ("") created 1113840807.028

    DO_MSLEEP(16);
    CtxPoint(309, 208);  //1113840807.326

    // Window CWI_12 ("Microsoft Word") created 1113840807.326

//////////////////////////////////////////////////////////////////////
/// The CtxSetWindowMatchTitle command is inserted prior to the
/// CtxSetWindowMatchTitle call. Note that the window identifier is the
//  same for both calls, and the "match pattern" for the command is from
/// Example Two in the Match Pattern Identification section above.
//////////////////////////////////////////////////////////////////////

    CtxSetWindowMatchTitle(CW1_12, "Sample Application ??:??:?? ?M" );

/// The window create wait event call does not change.
    CtxWaitForWindowCreate(CWI_12, 266);

    DO_MSLEEP(46);
    CWI_12->setTitle("Document1 - Microsoft Word"); //1113840807.388

    DO_MSLEEP(16);
    // Window CWI_11 ("") destroyed 1113840807.388

    DO_MSLEEP(4188);
    CtxPoint(626, 8);  //1113840811.693

    DO_MSLEEP(109);
    CtxClick(CWI_12, 94, L_BUTTON, NONE); //1113840811.787

    DO_MSLEEP(218);
    CtxPoint(623, 40);  //1113840812.006

    // Window CWI_12 ("Document1 - Microsoft Word") destroyed 1113840812.006
```

```
    DO_MSLEEP(2204);
    CtxPoint(385, 1);  //1113840815.262

    DO_MSLEEP(1046);
    // Window CWI_9 ("Program Manager") destroyed 1113840815.262

    // Window CWI_7 ("") destroyed 1113840815.262

    // Window CWI_13 ("Please wait...") created 1113840815.513

    CtxWaitForWindowCreate(CWI_13, 0);

    DO_SetTransactionCleanup();

    CtxDisconnect();

    END_TRANSACTION();
    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "Citrix License Warning Notice"
    delete CWI_6; // "UsrLogon.Cmd"
    delete CWI_7; // ""
    delete CWI_8; // "ICA Seamless Host Agent"
    delete CWI_9; // "Program Manager"
    delete CWI_10; // ""
    delete CWI_11; // ""
    delete CWI_12; // "Microsoft Word"
    delete CWI_13; // "Please wait..."

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Handling Intermittent Windows

Modifying the Script for Intermittent Windows

Windows that don't appear when a script is recorded can appear intermittently during replay. One example commonly encountered with Citrix is the ICA Seamless Host Agent window. If an unexpected window appears at validation or playback, you must modify the script to handle the window event.

To simplify the scripting process, record a temporary session and convert it to a script. This should be a session where the unexpected window appears so that the user must interact, for example, with a mouse click or keyboard entry, to dismiss the intermittent window. Note the location in the playback script where the presence of the intermittent window prevented the script from continuing. This is where code is added to the script.

Do the following if a validation or playback session indicates an unexpected window appeared that requires user interaction:

- Record a temporary script where the unexpected window event appears.
- Convert the original session to a script.
- Modify the original script with a section of the temporary script.

To record a temporary script:

Simplify the scripting effort required by doing the following:

1. Record a temporary session of the transaction. This should include the appearance of the intermittent window and the subsequent user interaction that dismisses the window.
2. Add a comment when the window appears and before the window is dismissed.
3. Give the successful record session a temporary name.
4. Click Options>Convert, and click OK to convert the session to a script.

To modify the original script:

Extract a small section of the temporary script code and insert it into the original script.

1. Identify the location in the original script where the unexpected window appeared. You can do this by noting the last window that was successfully created before validation failure.

Tip: Note the location with a code-style comment. This is the location where you paste in code from the temporary script.

2. Identify the code in the temporary script that creates the Citrix window object. This code is in the section labeled /* Citrix Window Information Objects */ and can be identified by the name parameter.
3. Cut and paste this line into the corresponding section in the original script.
4. Modify the line pasted into the original script, giving the statement a unique Citrix window identifier.

Note: This identifier, CWI_n, must be a unique value in the original script or the script will not compile.

5. Find the line in the temporary script that deletes the Citrix window object. This code is after the END_TRANSACTION call.
6. Copy this line to the same location in the original script and modify it with the unique Citrix window identifier from Step 4.
7. Add a special version of the CtxSetWindowMatchTitle command in the original script in the location where the original script failed because of the intermittent window.

   This is where the window must be recognized and dismissed, if it exists. The first parameter is the Citrix window object identifier from Step 2. The second parameter is an asterisk enclosed in double quotes (“*”). This parameter ensures that commands like CtxClick work with any matched window, even if the intermittent window does not exist.

9. Identify the code that dismisses this window in the temporary script by scrolling to the comment you inserted during capture, and backtracking until you find the correct WaitForWindowCreate statement. Usually this code consists of either a set of CtxPoint and CtxClick or one or more keyboard entry calls after the window create event.

Caution: Do not include the WaitForWindowCreate statement.

10. Copy the code from the temporary script and paste it after the CtxSetWindowMatchTitle call added in Step 5. Ensure that the Citrix window object parameter for these calls is the Citrix window object identifier from Step 2.

Conclusion

Following these techniques, you can modify a session to handle the appearance of intermittent windows that require user action to dismiss. The sample original script, the sample temporary script, and the sample modified script illustrate this process.

Sample: Original Citrix Script for Intermittent Windows

The following is an example of an original Citrix script converted from capture. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  intermittent_original.cpp
 *
 *  Script Converted on April 18, 2005 at 03:23:41 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                          : 80 characters
 *  Sleep Seconds                       : 1
 *  Auto Checkpoints                    : No
 * Citrix
 *  General Options                     :
 *   Replay Output Mode                 : Normal
 *   Enable Counters                    : No
 *  Timeout Value Options               :
 *    Connect Timeout (s)               : 60
 *    Disconnect Timeout (s)            : 60
 *    Window Creation Timeout (s)       : 30
 *    Ping Timeout (s)                  : 20
 *    Wait Point Timeout (s)            : 30
 *  Input Options                       :
 *   Combine Keyboard Input             : Yes
 *   Combine Mouse Input                : Yes
 *  Window Options                      :
 *   Window Verification                : Yes
 *    Window Max Retries                : 5
 *    Window Wait Retries (ms)          : 5000
 *   Enable Wildcard Title Match        : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
```

```
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_9 = new CtxWI(0x100b8, "", 115, 0, 405, 457);
   CtxWI *CWI_10 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);
/// A line will be copied and modified here to create the new Citrix window object

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("intermittent_original.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
   CtxSetEnableCounters(FALSE);
   CtxSetWindowRetries(5, 5000);
   CtxSetEnableWildcardMatching(TRUE);

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   CtxConnect(CitrixServer, CitrixOutputMode);

   // Window CWI_1 ("Warning !!") created 1113852175.329

   CtxWaitForWindowCreate(CWI_1, 2047);

   DO_MSLEEP(1203);
   CtxPoint(162, 161);  //1113852176.561

   DO_MSLEEP(31);
   CtxClick(CWI_1, 79, L_BUTTON, NONE); //1113852176.639

   DO_MSLEEP(718);
   CtxPoint(323, 362);  //1113852177.451

   DO_MSLEEP(94);
   CtxClick(CWI_1, 63, L_BUTTON, NONE); //1113852177.513

   // Window CWI_2 ("Log On to Windows") created 1113852177.638

   CtxWaitForWindowCreate(CWI_2, 125);

   // Window CWI_1 ("Warning !!") destroyed 1113852177.638
```

```
CtxType(CWI_2, "citr");  //1113852180.291

DO_MSLEEP(2625);
CtxType(CWI_2, "ix");  //1113852180.790


DO_MSLEEP(343);
CtxPoint(324, 362);  //1113852180.790

CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113852180.868

DO_MSLEEP(266);
CtxType(CWI_2, "citrix");  //1113852182.194

DO_MSLEEP(1078);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113852182.225

DO_MSLEEP(281);
CtxType(CWI_2, "q");  //1113852183.692

DO_MSLEEP(1078);

DO_MSLEEP(844);
CtxPoint(263, 324);  //1113852184.176

DO_MSLEEP(31);
CtxClick(CWI_2, 79, L_BUTTON, NONE); //1113852184.254

DO_MSLEEP(109);
// Window CWI_2 ("Log On to Windows") destroyed 1113852184.363

// Window CWI_3 ("Please wait...") created 1113852184.394

CtxWaitForWindowCreate(CWI_3, 16);

// Window CWI_4 ("Citrix License Warning Notice") created 1113852184.457

CtxWaitForWindowCreate(CWI_4, 78);

// Window CWI_3 ("Please wait...") destroyed 1113852184.472

// Window CWI_5 ("Citrix License Warning Notice") created 1113852184.566

CtxWaitForWindowCreate(CWI_5, 109);

// Window CWI_6 ("UsrLogon.Cmd") created 1113852184.566

CtxWaitForWindowCreate(CWI_6, 0);

// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113852184.566

DO_MSLEEP(31);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113852184.597

// Window CWI_7 ("") created 1113852184.878

// Window CWI_8 ("Program Manager") created 1113852184.941

CtxWaitForWindowCreate(CWI_8, 0);

CtxPoint(195, 192);  //1113852186.891

// Window CWI_9 ("") created 1113852186.891
```

```
    DO_MSLEEP(516);
    CtxPoint(316, 271);  //1113852187.406

    CtxMouseDown(CWI_5, L_BUTTON, NONE, 316, 271);  // 1113852187.406

    CtxMouseUp(CWI_5, L_BUTTON, NONE, 317, 278); //1113852187.515

    DO_MSLEEP(219);
    CtxPoint(317, 287);  //1113852187.671

    DO_MSLEEP(46);
    CtxClick(CWI_5, 63, L_BUTTON, NONE); //1113852187.734

    DO_MSLEEP(62);
    // Window CWI_5 ("Citrix License Warning Notice") destroyed 1113852187.796

/// Below is the last statement that executed prior to script failure
/// This is where the code to identify and dismiss the window will be added.

    DO_MSLEEP(829);
    CtxPoint(555, 200);  //1113852188.639

    DO_MSLEEP(15);
    CtxClick(CWI_9, 94, L_BUTTON, NONE); //1113852188.732

    DO_MSLEEP(875);
    CtxPoint(555, 199);  //1113852189.934

    DO_MSLEEP(328);
    // Window CWI_9 ("") destroyed 1113852189.934

    DO_MSLEEP(19766);
    CtxPoint(337, 157);  //1113852210.000

    DO_MSLEEP(328);
    // Window CWI_9 ("Program Manager") destroyed 1113852210.000

    // Window CWI_7 ("") destroyed 1113852210.016

    // Window CWI_10 ("Please wait...") created 1113852210.266

    CtxWaitForWindowCreate(CWI_10, 0);

    DO_SetTransactionCleanup();

    CtxDisconnect();


    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "Citrix License Warning Notice"
    delete CWI_6; // "UsrLogon.Cmd"
    delete CWI_7; // ""
    delete CWI_8; // "Program Manager"
    delete CWI_9; // ""
    delete CWI_10; // "Please wait..."
/// A line will be copied here and modified to delete the Citrix window object.

    CitrixUninit();

    EXIT();
```

```
   return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");

   CitrixUninit();

   EXIT();
}
```

Sample: Modified Citrix Script to Handle Intermittent Window

The following is an example of a modified Citrix script to handle an intermittent window. Changes to Original script are highlighted in bold.

Sample Script

```
/*
 *  intermittent_original.cpp
 *
 *  Script Converted on April 18, 2005 at 03:23:41 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                     : 80 characters
 *  Sleep Seconds                  : 1
 *  Auto Checkpoints               : No
 * Citrix
 *  General Options                :
 *   Replay Output Mode            : Normal
 *   Enable Counters               : No
 *  Timeout Value Options          :
 *    Connect Timeout (s)          : 60
 *    Disconnect Timeout (s)       : 60
 *    Window Creation Timeout (s)  : 30
 *    Ping Timeout (s)             : 20
 *    Wait Point Timeout (s)       : 30
 *  Input Options                  :
 *   Combine Keyboard Input        : Yes
 *   Combine Mouse Input           : Yes
 *  Window Options                 :
 *   Window Verification           : Yes
 *    Window Max Retries           : 5
 *    Window Wait Retries (ms)     : 5000
 *   Enable Wildcard Title Match   : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
```

```
#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_8 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_9 = new CtxWI(0x100b8, "", 115, 0, 405, 457);
   CtxWI *CWI_10 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);
/// This line was added from the temporary script and modified so the Citrix window variable
is unique for the original script.
   CtxWI *CWI_99 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("intermittent_original.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
   CtxSetEnableCounters(FALSE);
   CtxSetWindowRetries(5, 5000);
   CtxSetEnableWildcardMatching(TRUE);

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   CtxConnect(CitrixServer, CitrixOutputMode);

   // Window CWI_1 ("Warning !!") created 1113852175.329

   CtxWaitForWindowCreate(CWI_1, 2047);

   DO_MSLEEP(1203);
   CtxPoint(162, 161);  //1113852176.561

   DO_MSLEEP(31);
   CtxClick(CWI_1, 79, L_BUTTON, NONE); //1113852176.639

   DO_MSLEEP(718);
   CtxPoint(323, 362);  //1113852177.451

   DO_MSLEEP(94);
```

```
CtxClick(CWI_1, 63, L_BUTTON, NONE); //1113852177.513

// Window CWI_2 ("Log On to Windows") created 1113852177.638

CtxWaitForWindowCreate(CWI_2, 125);

// Window CWI_1 ("Warning !!") destroyed 1113852177.638

CtxType(CWI_2, "citr");  //1113852180.291

DO_MSLEEP(2625);
CtxType(CWI_2, "ix");  //1113852180.790

DO_MSLEEP(343);
CtxPoint(324, 362);  //1113852180.790

CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113852180.868

DO_MSLEEP(266);
CtxType(CWI_2, "citrix");  //1113852182.194

DO_MSLEEP(1078);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113852182.225

DO_MSLEEP(281);
CtxType(CWI_2, "q");  //1113852183.692

DO_MSLEEP(1078);

DO_MSLEEP(844);
CtxPoint(263, 324);  //1113852184.176

DO_MSLEEP(31);
CtxClick(CWI_2, 79, L_BUTTON, NONE); //1113852184.254

DO_MSLEEP(109);
// Window CWI_2 ("Log On to Windows") destroyed 1113852184.363

// Window CWI_3 ("Please wait...") created 1113852184.394

CtxWaitForWindowCreate(CWI_3, 16);

// Window CWI_4 ("Citrix License Warning Notice") created 1113852184.457

CtxWaitForWindowCreate(CWI_4, 78);

// Window CWI_3 ("Please wait...") destroyed 1113852184.472

// Window CWI_5 ("Citrix License Warning Notice") created 1113852184.566

CtxWaitForWindowCreate(CWI_5, 109);

// Window CWI_6 ("UsrLogon.Cmd") created 1113852184.566

CtxWaitForWindowCreate(CWI_6, 0);

// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113852184.566

DO_MSLEEP(31);
// Window CWI_6 ("UsrLogon.Cmd") destroyed 1113852184.597

// Window CWI_7 ("") created 1113852184.878

DO_MSLEEP(282);
```

```
    // Window CWI_9 ("Program Manager") created 1113852184.941

    CtxWaitForWindowCreate(CWI_8, 0);

    CtxPoint(195, 192);  //1113852186.891

    // Window CWI_10 ("") created 1113852186.891

    DO_MSLEEP(516);
    CtxPoint(316, 271);  //1113852187.406

    CtxMouseDown(CWI_5, L_BUTTON, NONE, 316, 271);  // 1113852187.406

    CtxMouseUp(CWI_5, L_BUTTON, NONE, 317, 278); //1113852187.515

    DO_MSLEEP(219);
    CtxPoint(317, 287);  //1113852187.671

    DO_MSLEEP(46);
    CtxClick(CWI_5, 63, L_BUTTON, NONE); //1113852187.734

    DO_MSLEEP(62);
    // Window CWI_5 ("Citrix License Warning Notice") destroyed 1113852187.796

/// This is the location where the code to set the intermittent
/// window match name is added.

    CtxSetWindowMatchTitle( CWI_99, "*" );

/// The location where the code to dismiss the code was copied from
/// the temporary script.  The parameter for the CtxClick command
/// was corrected with the Citrix window object variable.

    DO_MSLEEP(14000);
    CtxPoint(178, 194);  //1113851799.788

    DO_MSLEEP(47);
    CtxClick(CWI_8, 63, L_BUTTON, NONE); //1113851799.866

    DO_MSLEEP(829);
    CtxPoint(555, 200);  //1113852188.639

    DO_MSLEEP(15);
    CtxClick(CWI_9, 94, L_BUTTON, NONE); //1113852188.732

    DO_MSLEEP(875);
    CtxPoint(555, 199);  //1113852189.934

    DO_MSLEEP(328);
    // Window CWI_10 ("") destroyed 1113852189.934

    DO_MSLEEP(19766);
    CtxPoint(337, 157);  //1113852210.000

    DO_MSLEEP(328);
    // Window CWI_9 ("Program Manager") destroyed 1113852210.000

    // Window CWI_7 ("") destroyed 1113852210.016

    // Window CWI_11 ("Please wait...") created 1113852210.266

    CtxWaitForWindowCreate(CWI_10, 0);

    DO_SetTransactionCleanup();
```

```
   CtxDisconnect();

   END_TRANSACTION();

   delete CWI_1; // "Warning !!"
   delete CWI_2; // "Log On to Windows"
   delete CWI_3; // "Please wait..."
   delete CWI_4; // "Citrix License Warning Notice"
   delete CWI_5; // "Citrix License Warning Notice"
   delete CWI_6; // "UsrLogon.Cmd"
   delete CWI_7; // ""
   delete CWI_8; // "ICA Seamless Host Agent"
   delete CWI_9; // "Program Manager"
   delete CWI_10; // ""
   delete CWI_11; // "Please wait..."
/// The code copied and modified to delete the Citrix window object.
   delete CWI_99;

   CitrixUninit();

   EXIT();
   return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");

   CitrixUninit();

   EXIT();
}
```

Sample: Temporary Citrix Script with Window Creation and Dismissal

The following is an example of a temporary Citrix script with window creation and dismissal. Code to be identified and copied from the temporary script is highlighted in bold.

Sample Script

```
/*
 *  intermittent_temporary.cpp
 *
 *  Script Converted on April 18, 2005 at 03:16:57 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : No
 * Citrix
 *  General Options                 :
 *   Replay Output Mode             : Normal
 *   Enable Counters                : No
 *  Timeout Value Options           :
 *    Connect Timeout (s)           : 60
 *    Disconnect Timeout (s)        : 60
 *    Window Creation Timeout (s)   : 30
 *    Ping Timeout (s)              : 20
 *    Wait Point Timeout (s)        : 30
```

```
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
   CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
/// The code line to copy to the original script.
   CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_10 = new CtxWI(0x10084, "", 115, 0, 405, 457);

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("intermittent_temporary.cpp");

   CitrixInit(1);

   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
   CtxSetEnableCounters(FALSE);
   CtxSetWindowRetries(5, 5000);
   CtxSetEnableWildcardMatching(TRUE);

   SYNCHRONIZE();

   BEGIN_TRANSACTION();
```

```
DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

CtxPoint(120, 159);  //1113851768.564

// Window CWI_1 ("Warning !!") created 1113851768.564

CtxWaitForWindowCreate(CWI_1, 2063);

DO_MSLEEP(766);
CtxPoint(313, 361);  //1113851769.344

DO_MSLEEP(31);
CtxClick(CWI_1, 62, L_BUTTON, NONE); //1113851769.407

// Window CWI_2 ("Log On to Windows") created 1113851769.454

CtxWaitForWindowCreate(CWI_2, 47);

// Window CWI_1 ("Warning !!") destroyed 1113851769.454

CtxType(CWI_2, "citrix");  //1113851774.556

DO_MSLEEP(4891);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113851774.618

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113851776.460

DO_MSLEEP(1594);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113851776.522

DO_MSLEEP(312);
CtxType(CWI_2, "q");  //1113851779.627

DO_MSLEEP(2469);

DO_MSLEEP(953);
CtxPoint(231, 322);  //1113851780.017

DO_MSLEEP(78);
CtxMouseDown(CWI_2, L_BUTTON, NONE, 231, 322);  // 1113851780.017

CtxMouseUp(CWI_2, L_BUTTON, NONE, 231, 321); //1113851780.080

DO_MSLEEP(16);
CtxPoint(231, 321);  //1113851780.127

DO_MSLEEP(94);
// Window CWI_2 ("Log On to Windows") destroyed 1113851780.127

// Window CWI_3 ("Please wait...") created 1113851780.236

CtxWaitForWindowCreate(CWI_3, 109);

// Window CWI_4 ("Citrix License Warning Notice") created 1113851780.298

CtxWaitForWindowCreate(CWI_4, 63);

// Window CWI_3 ("Please wait...") destroyed 1113851780.298

// Window CWI_5 ("Citrix License Warning Notice") created 1113851780.314

CtxWaitForWindowCreate(CWI_5, 15);
```

```
    // Window CWI_6 ("UsrLogon.Cmd") created 1113851780.330

    CtxWaitForWindowCreate(CWI_6, 16);

    // Window CWI_4 ("Citrix License Warning Notice") destroyed 1113851780.330

    DO_MSLEEP(78);
    // Window CWI_6 ("UsrLogon.Cmd") destroyed 1113851780.408

    // Window CWI_7 ("") created 1113851780.766

    DO_MSLEEP(344);
    // Window CWI_8 ("ICA Seamless Host Agent") created 1113851780.766

    CtxWaitForWindowCreate(CWI_8, 15);

    // Window CWI_9 ("Program Manager") created 1113851780.766

    CtxWaitForWindowCreate(CWI_9, 0);

    DO_MSLEEP(891);
    CtxPoint(333, 300);  //1113851781.718

    DO_MSLEEP(62);
    CtxClick(CWI_5, 141, L_BUTTON, NONE); //1113851781.859

    DO_MSLEEP(156);
    CtxPoint(261, 250);  //1113851782.186

    DO_MSLEEP(172);
    // Window CWI_5 ("Citrix License Warning Notice") destroyed 1113851782.186

    CtxPoint(303, 251);  //1113851782.811

    // Window CWI_10 ("") created 1113851782.811

    DO_MSLEEP(625);

    DO_MSLEEP(2078);
    CtxPoint(518, 254);  //1113851785.760

    DO_MSLEEP(875);
    // Window CWI_10 ("") destroyed 1113851785.760
/// The following code dismisses the “ICA Seamless Host Agent”.
/// Copy the bold section below to the original script.
/// The DO_MSLEEP comments can also be copied.

    DO_MSLEEP(14000);
    CtxPoint(178, 194);  //1113851799.788

    DO_MSLEEP(47);
    CtxClick(CWI_8, 63, L_BUTTON, NONE); //1113851799.866

/// End of code section to copy.

    DO_MSLEEP(125);
    CtxPoint(183, 197);  //1113851799.975

    // Window CWI_8 ("ICA Seamless Host Agent") destroyed 1113851799.975

    DO_MSLEEP(1250);
    CtxPoint(609, 12);  //1113851804.469

    DO_MSLEEP(3250);
```

```
    // Window CWI_9 ("Program Manager") destroyed 1113851804.469

    // Window CWI_7 ("") destroyed 1113851804.469

    DO_SetTransactionCleanup();

    CtxDisconnect();

    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "Citrix License Warning Notice"
    delete CWI_6; // "UsrLogon.Cmd"
    delete CWI_7; // ""
/// Copy this code to the original script to delete the Citrix window object.
    delete CWI_8; // "ICA Seamless Host Agent"
    delete CWI_9; // "Program Manager"
    delete CWI_10; // ""

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Handling Unexpected Events

Modifying the Script to Handle Unexpected Events

Unexpected Citrix events that were not recorded in the original script can occur during a playback session . These can include Intermittent Windows or application windows that may appear based on the user's session state. For example, the calculator application may already be present when the Citrix user logs on a session, or the user may need to invoke the application.

When there is the possibility of unexpected events occurring, you must modify the script to respond to the changes and continue the load test. Use the CtxWindowEventExists function to create a conditional block of code that handles the unexpected dialogs.

When you modify a script to handle unexpected events, you must:

- Perform an initial validation of the script.
- Record a temporary script to capture the unexpected event.
- Modify the original script to include and handle the unexpected event.
- Re-validate the script.

Note: You may need to configure the workbench and player for validation.

To perform an initial validation:

1. Click **Session>Validate** to validate the script. This is when an unexpected event can cause sporadic failure during validation or playback.
2. Identify the sporadic event visually so that you can recognize it in a later record session.

To record a temporary script:

1. Click **Options>Record** to record a second script that recreates the unexpected event. Successive sessions should be recorded until the unexpected event occurs.
2. Insert comments at two points:
   - Where the event is first recognized.
   - Where the event is acted on and the Citrix session state and any window states have returned to the state before the event occurred.

Note: You may need to position the mouse over a window or control, such as a button, before inserting the second comment. (See Scripting Mouse Events.)

3. Give this capture a name to denote this session's temporary status.
4. Click **Options>Convert**, then click **OK** to convert the session to a script.

To modify the original script with code from the temporary script:

Follow these steps to insert script code from the temporary script that handles the unexpected event.

1. Insert a code comment in the original script where the unexpected event occurred.
2. Copy the code between the two comments in the temporary script and paste it into the original script at the location of the unexpected event you identified.

   Note: Comments ensure that the pasted code is clearly marked in the script.

3. Identify any Citrix window objects that exist in API calls in the pasted code snippet. For each object, do one of the following:
   - If the window object is wholly contained, both created and destroyed, in the pasted code snippet:

   a. Identify the Citrix window creation and deletion lines of code in the temporary script.

   b. Copy these lines to the original script. This is the creation line in the /* Citrix Window Information Objects */ section and the deletion line after the END_TRANSACTION call.

   c. Give the Citrix window object variable a unique variable name.

   d. Change all variable references to the Citrix window in the pasted code snippet API calls to the new variable name.

   - If the window object referenced in a pasted API call is an action on an existing window object in the original script, modify all API calls in the pasted calls to refer to the variable name of the Citrix window object in the original script.
4. Before the code snippet, add a conditional check to see if the unexpected window event has occurred. Use the CtxWindowEventExists API call in an `If` conditional, where if the result is TRUE, a block of code is executed. Then add a BeginBlock on the next line. BeginBlock is logically identical to the C begin brace "{".
5. After the code snippet, add an EndBlock. EndBlock is logically identical to the C end brace "}".

   Note: If there is code in the original script that should not be executed if the condition is TRUE, put this code in an Else block within the BeginBlock and EndBlock calls.

To re-validate the script:

1. Click **Session>Compile** to compile the script after making scripting changes.

2. Click **Session>Validate Script** to validate the script. Make sure that validation succeeds when the unexpected event occurs and when the event does not occur.

Conclusion

Following these techniques, you can modify scripts to handle unexpected events that occur during playback. Scripting around unexpected events allows you to perform load testing for complex user scenarios. The sample original script and sample modified script illustrate this process.

Sample: Original Citrix Scripts for Unexpected Events

The following sample shows two examples of an unexpected Citrix event. In the first example, the calculator application must be started in the original script. The second example has a session where the calculator application is already present when the user logs on. Points of interest in the script are highlighted in bold.

Example 1: Original Record Script with No Calculator Application Present at Logon

```
/*
 *  CalculatorCreateNew.cpp
 *
 *  Script Converted on May 17, 2004 at 12:33:36 PM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 33
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                         : 80 characters
 *  Sleep Seconds                      : 1
 *  Auto Checkpoints                   : No
 * Citrix
 *  General Options                    :
 *   Window Verification               : Yes
 *   Session Timeouts                  : Yes
 *    Connect Timeout (s)              : 60
 *    Disconnect Timeout (s)           : 60
 *    Window Creation Timeout (s)      : 30
 *    Ping Timeout (s)                 : 20
 *    Wait Point Timeout (s)           : 30
 *   Include Wait Points               : Yes
 *   Enable Counters                   : No
 *   Include Unnamed Windows           : Yes
 *   Output Mode                       : Normal
 *  Input Options                      :
 *   Combine Keyboard Input            : Yes
 *   Combine Mouse Input               : Yes
 */
#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000205UL
#include <stdio.h>
#include "smacro.h"
#include "do_citrix.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
```

```
/* Declare Variables */
const char *CitrixServer      = "qaccitrix2";
const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

/* Citrix Window Information Objects */
CtxWI *CWI_1 = new CtxWI(0x1001a, "Warning !!", 107, 43, 427, 351);
CtxWI *CWI_2 = new CtxWI(0x2001a, "Log On to Windows", 111, 65, 418, 285);
CtxWI *CWI_3 = new CtxWI(0x3002e, "Please wait...", 111, 112, 418, 145);
CtxWI *CWI_4 = new CtxWI(0x40030, "UsrLogon.Cmd", 0, 456, 161, 25);
CtxWI *CWI_5 = new CtxWI(0x3002c, "", 0, 0, 641, 481);
CtxWI *CWI_6 = new CtxWI(0x20026, "", -2, 452, 645, 31);
CtxWI *CWI_7 = new CtxWI(0x10052, "ICA Seamless Host Agent", 0, 0, 391, 224);
CtxWI *CWI_8 = new CtxWI(0x10048, "Program Manager", 0, 0, 641, 481);
CtxWI *CWI_9 = new CtxWI(0x10084, "", 115, 0, 405, 457);
CtxWI *CWI_10 = new CtxWI(0x20058, "Calculator", 44, 44, 261, 255);
CtxWI *CWI_11 = new CtxWI(0x5002e, "Please wait...", 111, 112, 418, 145);

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("CalculatorCreateNew.cpp");
CitrixInit(1);
/* Citrix replay settings */
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
BEGIN_TRANSACTION();
CtxConnect(CitrixServer, CitrixOutputMode);

DO_MSLEEP(1172);
CtxPoint(172, 214);

DO_MSLEEP(250);
// Window CWI_1 ("Warning !!") created 1084811574.696

CtxWaitForWindowCreate(CWI_1);

DO_MSLEEP(906);
CtxPoint(308, 359);

DO_MSLEEP(156);
CtxClick(CWI_1, 78, L_BUTTON, NONE);

DO_MSLEEP(63);
// Window CWI_2 ("Log On to Windows") created 1084811575.899

CtxWaitForWindowCreate(CWI_2);

// Window CWI_1 ("Warning !!") destroyed 1084811575.899

Type(CWI_2, "citrix");

DO_MSLEEP(2547);
TypeVK(CWI_2, VK_TAB, NONE);

DO_MSLEEP(328);
Type(CWI_2, "citrix");

DO_MSLEEP(3015);
```

```
TypeVK(CWI_2, VK_TAB, NONE);

DO_MSLEEP(282);
Type(CWI_2, "q");

DO_MSLEEP(703);

DO_MSLEEP(1343);
CtxPoint(247, 322);

DO_MSLEEP(110);
CtxClick(CWI_2, 62, L_BUTTON, NONE);

DO_MSLEEP(391);
// Window CWI_2 ("Log On to Windows") destroyed 1084811584.685

DO_MSLEEP(16);
// Window CWI_3 ("Please wait...") created 1084811584.701

CtxWaitForWindowCreate(CWI_3);

DO_MSLEEP(156);
// Window CWI_3 ("Please wait...") destroyed 1084811584.857

DO_MSLEEP(156);
// Window CWI_4 ("UsrLogon.Cmd") created 1084811585.013

CtxWaitForWindowCreate(CWI_4);

DO_MSLEEP(63);
CtxPoint(259, 322);

// Window CWI_4 ("UsrLogon.Cmd") destroyed 1084811585.076

DO_MSLEEP(187);
CtxPoint(449, 462);

DO_MSLEEP(63);
// Window CWI_5 ("") created 1084811585.326

DO_MSLEEP(93);
// Window CWI_5 ("") destroyed 1084811585.420

DO_MSLEEP(16);
// Window CWI_6 ("") created 1084811585.436

DO_MSLEEP(219);
// Window CWI_7 ("ICA Seamless Host Agent") created 1084811585.654

CtxWaitForWindowCreate(CWI_7);

DO_MSLEEP(15);
// Window CWI_8 ("Program Manager") created 1084811585.670

CtxWaitForWindowCreate(CWI_8);

DO_MSLEEP(641);
CtxPoint(135, 347);

DO_MSLEEP(3359);
// Window CWI_9 ("") created 1084811589.672

DO_MSLEEP(3032);
CtxPoint(192, 216);
```

```
    // Window CWI_9 ("") destroyed 1084811592.705

    DO_MSLEEP(921);
    CtxPoint(190, 199);

    DO_MSLEEP(63);
    CtxClick(CWI_6, 78, L_BUTTON, NONE);

    DO_MSLEEP(16);
    // Window CWI_7 ("ICA Seamless Host Agent") destroyed 1084811593.784

/// These methods are, by placement in the script prior to the creation
/// of the calculator window, deduced to be the actions that result in the
/// creation of the Calculator application window. Paying attention to actions
/// performed while recording makes this task easier.
    DO_MSLEEP(3843);
    CtxPoint(112, 98);

    DO_MSLEEP(282);
    CtxDoubleClick(CWI_8);

    DO_MSLEEP(110);

    DO_MSLEEP(140);
    // Window CWI_10 ("Calculator") created 1084811598.255

/// Scripting Step 1:
/// The calculator window is created at this point
/// Working backward, we deduce that the CtxPoint and
/// CtxDoubleClick API calls created this window.
    CtxWaitForWindowCreate(CWI_10);

    DO_MSLEEP(2672);
    CtxPoint(295, 48);

    DO_MSLEEP(672);
    CtxClick(CWI_10, 94, L_BUTTON, NONE);

    // Window CWI_10 ("Calculator") destroyed 1084811601.694

    DO_MSLEEP(1125);
    CtxPoint(256, 95);

    DO_MSLEEP(281);
    // Window CWI_8 ("Program Manager") destroyed 1084811603.101

    // Window CWI_6 ("") destroyed 1084811603.101

    // Window CWI_11 ("Please wait...") created 1084811603.601

    CtxWaitForWindowCreate(CWI_11);

    CtxDisconnect();

    END_TRANSACTION();
    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "UsrLogon.Cmd"
    delete CWI_5; // ""
    delete CWI_6; // ""
    delete CWI_7; // "ICA Seamless Host Agent"
    delete CWI_8; // "Program Manager"
    delete CWI_9; // ""
    delete CWI_10; // "Calculator"
```

```
     delete CWI_11; // "Please wait..."

     CitrixUninit();

     REPORT(SUCCESS);
     EXIT();
     return(0);
     }
void abort_function(PLAYER_INFO *s_info)
{
     RR__printf("Virtual User ABORTED.");

     CitrixUninit();

     EXIT();
}
```

Example 2: Original Record Script with Calculator Application Present at Logon

Points of interest in the script are highlighted in bold.

```
/*
 *  CalculatorUseExisting.cpp
 *
 *  Script Converted on May 17, 2004 at 12:35:29 PM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 33
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Citrix
 *  General Options                  :
 *   Window Verification             : Yes
 *   Session Timeouts                : Yes
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *   Include Wait Points             : Yes
 *   Enable Counters                 : No
 *   Include Unnamed Windows         : Yes
 *   Output Mode                     : Normal
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 */
#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000205UL
#include <stdio.h>
#include "smacro.h"
#include "do_citrix.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
     /* Declare Variables */
```

```
const char *CitrixServer      = "qaccitrix2";
const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

/* Citrix Window Information Objects */
CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
CtxWI *CWI_3 = new CtxWI(0x20026, "", -2, 740, 1029, 31);
CtxWI *CWI_4 = new CtxWI(0x10136, "Calculator", 154, 154, 261, 253);
CtxWI *CWI_5 = new CtxWI(0x10048, "Program Manager", 0, 0, 1025, 769);
CtxWI *CWI_6 = new CtxWI(0x2017c, "ICA Seamless Host Agent", 0, 0, 391, 224);
CtxWI *CWI_7 = new CtxWI(0x5002e, "Please wait...", 303, 208, 418, 145);

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("CalculatorUseExisting.cpp");
CitrixInit(1);
/* Citrix replay settings */
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
BEGIN_TRANSACTION();
CtxConnect(CitrixServer, CitrixOutputMode);

DO_MSLEEP(625);
CtxPoint(130, 78);

DO_MSLEEP(797);
// Window CWI_1 ("Warning !!") created 1084811688.065

CtxWaitForWindowCreate(CWI_1);

DO_MSLEEP(1813);
CtxPoint(309, 365);

DO_MSLEEP(203);
CtxClick(CWI_1, 140, L_BUTTON, NONE);

DO_MSLEEP(63);
// Window CWI_2 ("Log On to Windows") created 1084811690.285

CtxWaitForWindowCreate(CWI_2);

DO_MSLEEP(15);
// Window CWI_1 ("Warning !!") destroyed 1084811690.300

CtxType(CWI_2, "citrix");

DO_MSLEEP(2375);
CtxPoint(309, 364);

CtxTypeVK(CWI_2, VK_TAB, NONE);

DO_MSLEEP(813);
CtxType(CWI_2, "citrix");

DO_MSLEEP(1844);
CtxTypeVK(CWI_2, VK_TAB, NONE);
```

```
     DO_MSLEEP(328);
     CtxType(CWI_2, "q");

     DO_MSLEEP(672);

     DO_MSLEEP(1281);
     CtxPoint(241, 317);

     DO_MSLEEP(125);
     CtxClick(CWI_2, 141, L_BUTTON, NONE);

     DO_MSLEEP(406);
     // Window CWI_2 ("Log On to Windows") destroyed 1084811698.289

     DO_MSLEEP(828);
     CtxPoint(514, 358);

     DO_MSLEEP(125);
     // Window CWI_3 ("") created 1084811699.258

     DO_MSLEEP(16);
     // Window CWI_4 ("Calculator") created 1084811699.258

/// The following comment inserted during record indicates that the
/// calculator window has appeared in the script.
/// Since it happens before any events after the Logon window
/// was destroyed, we know that in this session, we will not have to
/// start the calculator application – it is already present!
     /*
     The calculator has been created at this point in the modified script!
     */
/// Scripting Step 2:
/// The code in bold is cut and pasted into the original script
     CtxWaitForWindowCreate(CWI_4);

     // Window CWI_5 ("Program Manager") created 1084811699.258

     CtxWaitForWindowCreate(CWI_5);

     DO_MSLEEP(640);
     // Window CWI_6 ("ICA Seamless Host Agent") created 1084811699.899

     CtxWaitForWindowCreate(CWI_6);

     DO_MSLEEP(1672);
     CtxPoint(200, 195);

     DO_MSLEEP(234);
     CtxClick(CWI_6, 141, L_BUTTON, NONE);

     DO_MSLEEP(16);
     // Window CWI_6 ("ICA Seamless Host Agent") destroyed 1084811701.962

     DO_MSLEEP(672);
     CtxPoint(200, 205);

     DO_MSLEEP(125);
     CtxClick(CWI_3, 109, L_BUTTON, NONE);

     DO_MSLEEP(719);
     CtxPoint(275, 316);

     DO_MSLEEP(31);
     CtxMouseDown(CWI_4, L_BUTTON, NONE, 275, 316);
```

```
      CtxMouseUp(CWI_4, L_BUTTON, NONE, 274, 316);

      DO_MSLEEP(812);
      CtxPoint(349, 376);

      DO_MSLEEP(172);
      CtxClick(CWI_4, 110, L_BUTTON, NONE);

      DO_MSLEEP(1297);
      CtxPoint(259, 313);

      DO_MSLEEP(46);
      CtxClick(CWI_4, 79, L_BUTTON, NONE);

      DO_MSLEEP(843);
      CtxPoint(387, 374);

      CtxMouseDown(CWI_4, L_BUTTON, NONE, 387, 374);

      CtxMouseUp(CWI_4, L_BUTTON, NONE, 387, 375);

      DO_MSLEEP(3782);
      CtxPoint(397, 160);

      DO_MSLEEP(1140);
      CtxClick(CWI_4, 125, L_BUTTON, NONE);

      DO_MSLEEP(16);
      // Window CWI_4 ("Calculator") destroyed 1084811712.045
/// Scripting Step 2:
/// This is the end of the code snippet that will be cut and pasted into the
/// original script.
/// The following comment inserted during record indicates that the
/// calculator window has been destroyed.
/// Since it happens before any events after the Logon window
/// was destroyed, we know that in this session, we will not have to
/// start the calculator application – it is already present!
      /*
      The calculator has been destroyed at this point in the modified script!
      */
      DO_MSLEEP(1469);
      CtxPoint(235, 85);

      DO_MSLEEP(343);
      // Window CWI_5 ("Program Manager") destroyed 1084811713.858

      // Window CWI_3 ("") destroyed 1084811713.858

      // Window CWI_7 ("Please wait...") created 1084811714.359

      CtxWaitForWindowCreate(CWI_7);

      CtxDisconnect();

      END_TRANSACTION();
      delete CWI_1; // "Warning !!"
      delete CWI_2; // "Log On to Windows"
      delete CWI_3; // ""
      delete CWI_4; // "Calculator"
      delete CWI_5; // "Program Manager"
      delete CWI_6; // "ICA Seamless Host Agent"
      delete CWI_7; // "Please wait..."

      CitrixUninit();
```

```
     REPORT(SUCCESS);
     EXIT();
     return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
     RR__printf("Virtual User ABORTED.");

     CitrixUninit();

     EXIT();
}
```

Sample: Modified Citrix Script for Handling Unexpected Events

The following sample shows a Citrix script that is modified to address whether the calculator application is present is present at logon. Changes to Original script are highlighted in bold.

Sample Script

```
/*
 *  CalculatorUseExisting.cpp
 *
 *  Script Converted on May 17, 2004 at 12:35:29 PM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 33
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Citrix
 *  General Options                  :
 *   Window Verification             : Yes
 *   Session Timeouts                : Yes
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *   Include Wait Points             : Yes
 *   Enable Counters                 : No
 *   Include Unnamed Windows         : Yes
 *   Output Mode                     : Normal
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 */
#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000205UL
#include <stdio.h>
#include "smacro.h"
#include "do_citrix.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
extern "C" int rhobot_script(PLAYER_INFO *s_info)
```

```
{
    /* Declare Variables */
    const char *CitrixServer      = "qaccitrix2";
    const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

    /* Citrix Window Information Objects */
    CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
    CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
    CtxWI *CWI_3 = new CtxWI(0x20026, "", -2, 740, 1029, 31);
    CtxWI *CWI_4 = new CtxWI(0x10136, "Calculator", 154, 154, 261, 253);
    CtxWI *CWI_5 = new CtxWI(0x10048, "Program Manager", 0, 0, 1025, 769);
    CtxWI *CWI_6 = new CtxWI(0x2017c, "ICA Seamless Host Agent", 0, 0, 391, 224);
    CtxWI *CWI_7 = new CtxWI(0x5002e, "Please wait...", 303, 208, 418, 145);

    SET_ABORT_FUNCTION(abort_function);
    DEFINE_TRANS_TYPE("CalculatorUseExisting.cpp");
    CitrixInit(1);
    /* Citrix replay settings */
    CtxSetConnectTimeout(60);
    CtxSetDisconnectTimeout(60);
    CtxSetWindowTimeout(30);
    CtxSetPingTimeout(20);
    CtxSetWaitPointTimeout(30);
    CtxSetWindowVerification(TRUE);
    CtxSetEnableCounters(FALSE);
    CtxSetWindowRetries(5, 5000);
    CtxSetEnableWildcardMatching(TRUE);

    SYNCHRONIZE();
    BEGIN_TRANSACTION();
    CtxConnect(CitrixServer, CitrixOutputMode);

    DO_MSLEEP(625);
    CtxPoint(130, 78);

    DO_MSLEEP(797);
    // Window CWI_1 ("Warning !!") created 1084811688.065

    CtxWaitForWindowCreate(CWI_1);

    DO_MSLEEP(1813);
    CtxPoint(309, 365);

    DO_MSLEEP(203);
    CtxClick(CWI_1, 140, L_BUTTON, NONE);

    DO_MSLEEP(63);
    // Window CWI_2 ("Log On to Windows") created 1084811690.285

    CtxWaitForWindowCreate(CWI_2);

    DO_MSLEEP(15);
    // Window CWI_1 ("Warning !!") destroyed 1084811690.300

    CtxType(CWI_2, "citrix");

    DO_MSLEEP(2375);
    CtxPoint(309, 364);

    CtxTypeVK(CWI_2, VK_TAB, NONE);

    DO_MSLEEP(813);
    CtxType(CWI_2, "citrix");

    DO_MSLEEP(1844);
```

```
     CtxTypeVK(CWI_2, VK_TAB, NONE);

     DO_MSLEEP(328);
     CtxType(CWI_2, "q");

     DO_MSLEEP(672);

     DO_MSLEEP(1281);
     CtxPoint(241, 317);

     DO_MSLEEP(125);
     CtxClick(CWI_2, 141, L_BUTTON, NONE);

     DO_MSLEEP(406);
     // Window CWI_2 ("Log On to Windows") destroyed 1084811698.289

     DO_MSLEEP(828);
     CtxPoint(514, 358);

     DO_MSLEEP(125);
     // Window CWI_3 ("") created 1084811699.258

     DO_MSLEEP(16);
     // Window CWI_4 ("Calculator") created 1084811699.258

/// Here is where we can determine that either the Calculator application
/// exists in this session or it needs to be created.

/// Scripting Step
     if(CtxWindowEventExists("WindowCreate", 4000, "Calculator", ""))

/// If the window exists, the block below will be
/// executed.

      BeginBlock();

/// Scripting Step 2:
/// The following code was cut and pasted from the second script.

/// Scripting Step 3:
/// All Citrix window object name references have been reconciled with the
/// correct Citrix window object definition.

        CtxWaitForWindowCreate(CWI_4);

        // Window CWI_5 ("Program Manager") created 1084811699.258

        CtxWaitForWindowCreate(CWI_5);

        DO_MSLEEP(640);
        // Window CWI_6 ("ICA Seamless Host Agent") created 1084811699.899

        WaitForWindowCreate(CWI_6);

        DO_MSLEEP(1672);
        CtxPoint(200, 195);

        DO_MSLEEP(234);
        CtxClick(CWI_6, 141, L_BUTTON, NONE);

        DO_MSLEEP(16);
        // Window CWI_6 ("ICA Seamless Host Agent") destroyed 1084811701.962

        DO_MSLEEP(672);
        CtxPoint(200, 205);
```

```
        DO_MSLEEP(125);
        CtxClick(CWI_3, 109, L_BUTTON, NONE);

        DO_MSLEEP(719);
        CtxPoint(275, 316);

        DO_MSLEEP(31);
        CtxMouseDown(CWI_4, L_BUTTON, NONE, 275, 316);

        CtxMouseUp(CWI_4, L_BUTTON, NONE, 274, 316);

        DO_MSLEEP(812);
        CtxPoint(349, 376);

        DO_MSLEEP(172);
        CtxClick(CWI_4, 110, L_BUTTON, NONE);

        DO_MSLEEP(1297);
        CtxPoint(259, 313);

        DO_MSLEEP(46);
        CtxClick(CWI_4, 79, L_BUTTON, NONE);

        DO_MSLEEP(843);
        CtxPoint(387, 374);

        CtxMouseDown(CWI_4, L_BUTTON, NONE, 387, 374);

        CtxMouseUp(CWI_4, L_BUTTON, NONE, 387, 375);

        DO_MSLEEP(3782);
        CtxPoint(397, 160);

        DO_MSLEEP(1140);
        CtxClick(CWI_4, 125, L_BUTTON, NONE);

/// Scripting Step 5:
/// Here is the end of the code snippet, so an EndBlock goes here.

    EndBlock()

/// Scripting Step 6:
/// The original behavior has been put into the else
/// section of the if conditional between BeginBlock
/// and EndBlock commands.

    else
    BeginBlock();

        DO_MSLEEP(1672);
        CtxPoint(200, 195);

        DO_MSLEEP(234);
        CtxClick(CWI_6, 141, L_BUTTON, NONE);

        DO_MSLEEP(3843);
        CtxPoint(112, 98);

        DO_MSLEEP(282);
        CtxDoubleClick(CWI_5);

        CtxWaitForWindowCreate(CWI_4);

        DO_MSLEEP(2672);
        CtxPoint(295, 48);
```

```
        DO_MSLEEP(672);
        CtxClick(CWI_4, 94, L_BUTTON, NONE);

    EndBlock();

    DO_MSLEEP(16);
    // Window CWI_4 ("Calculator") destroyed 1084811712.045

    DO_MSLEEP(1469);
    CtxPoint(235, 85);

    DO_MSLEEP(343);
    // Window CWI_5 ("Program Manager") destroyed 1084811713.858

    // Window CWI_3 ("") destroyed 1084811713.858

    // Window CWI_7 ("Please wait...") created 1084811714.359

    //CtxWaitForWindowCreate(CWI_7);

    CtxDisconnect();

    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // ""
    delete CWI_4; // "Calculator"
    delete CWI_5; // "Program Manager"
    delete CWI_6; // "ICA Seamless Host Agent"
    delete CWI_7; // "Please wait..."

    CitrixUninit();

    REPORT(SUCCESS);
    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Moving Citrix Connect and Disconnect Actions

Modifying the Script to Move Connect and Disconnect Outside the Transaction Loop

Load testing for Citrix may require you to create extended logon sessions, in which the user remains connected to the Citrix server between transactions. When this is necessary, you can move the connect and disconnect actions outside the transaction loop so that the script does not logon and off with each iteration through the loop.

To modify the script, you must:

- Modify the recording process to prepare for moving the connect and disconnect actions.
- Modify the script.

To modify the recording process:

Prepare for moving the connect and disconnect actions outside the transaction loop by performing the following steps during recording:

1. Insert a comment, such as "Logged in to Citrix" , after the Citrix logon but before any windows are opened or any applications are started.
2. Ensure that all application windows are closed before disconnecting from the Citrix session.
3. Insert a comment, such as "Ready to log off Citrix", before the Citrix logoff sequence is initiated. You must place the second comment after all applications have been closed but before the user logs off.

To modify the script:

1. Comment out the BEGIN_TRANSACTION and END_TRANSACTION calls.
2. Add new BEGIN_TRANSACTION and END_TRANSACTION calls where the comments were placed in Step 1 and Step 3 in the recording process described above.
3. Comment out the DO_SetTransactionStart and DO_SetTransactionCleanup calls.

Note: Commenting out the calls instead of deleting them allows you to determine the original location of these commands for debugging purposes.

Conclusion

When you use these techniques, you can script a set of actions so that the script does not log on and log off with each iteration through the loop.

Caution: You must take care when you perform this action. In the event of a failure, you must log off before using a transaction restart.

The sample original script and sample modified script illustrate this process.

Sample: Original Citrix Script for Modifying the Transaction Loop

The following sample is an original Citrix script converted from capture. Points of interest in the script are shown in bold.

Sample Script

```
/*
 *  capCtxMoveLoop.cpp
 *
 *  Script Converted on February 28, 2005 at 08:30:06 AM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 231
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                         : 80 characters
 *  Sleep Seconds                      : 1
 *  Auto Checkpoints                   : No
 * Citrix
 *  General Options                    :
 *   Replay Output Mode                : Normal
 *   Enable Counters                   : No
 *  Timeout Value Options              :
```

```
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */
#define CITRIX_CLIENT_VERSION "8.100.29670"
#define CITRIX_ICO_VERSION    "2.4"
#define SCRIPT_VER 0x00000505UL
#include <stdio.h>
#include "smacro.h"
#include "do_citrix.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "QACCitrix";
   const char *CitrixUsername    = "Citrix";
   const char *CitrixPassword    = "~encr~657E06726F697206";
   const char *CitrixDomain      = "QACCitrix";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

   /* Citrix Window Information Objects */
   CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
   CtxWI *CWI_2 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
   CtxWI *CWI_3 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
   CtxWI *CWI_4 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
   CtxWI *CWI_5 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
   CtxWI *CWI_6 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
   CtxWI *CWI_7 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
   CtxWI *CWI_8 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
   CtxWI *CWI_9 = new CtxWI(0x2006c, "Calculator", 66, 66, 261, 253);

   SET_ABORT_FUNCTION(abort_function);
   DEFINE_TRANS_TYPE("capCtxMoveLoop.cpp");
   CitrixInit(1);
   /* Citrix replay settings */
   CtxSetConnectTimeout(60);
   CtxSetDisconnectTimeout(60);
   CtxSetWindowTimeout(30);
   CtxSetPingTimeout(20);
   CtxSetWaitPointTimeout(30);
   CtxSetWindowVerification(TRUE);
   CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, CitrixDomain);
   CtxSetEnableCounters(FALSE);
   CtxSetWindowRetries(5, 5000);
   CtxSetEnableWildcardMatching(TRUE);

   SYNCHRONIZE();
   BEGIN_TRANSACTION();
   DO_SetTransactionStart();

   CtxConnect(CitrixServer, CitrixOutputMode);
```

```
// Window CWI_1 ("Warning !!") created 1109597295.502

CtxWaitForWindowCreate(CWI_1, 2156);

DO_MSLEEP(922);
CtxPoint(333, 233);  //1109597296.611

DO_MSLEEP(188);
CtxMouseDown(CWI_1, L_BUTTON, NONE, 333, 233);  // 1109597296.611

CtxMouseUp(CWI_1, L_BUTTON, NONE, 332, 233); //1109597296.861

DO_MSLEEP(1250);
CtxPoint(319, 360);  //1109597297.938

DO_MSLEEP(78);
CtxMouseDown(CWI_1, L_BUTTON, NONE, 319, 360);  // 1109597297.938

CtxMouseUp(CWI_1, L_BUTTON, NONE, 320, 360); //1109597298.235

DO_MSLEEP(78);
CtxPoint(320, 360);  //1109597298.282

DO_MSLEEP(265);
// Window CWI_1 ("Warning !!") destroyed 1109597298.282

CtxPoint(321, 358);  //1109597298.469

// Window CWI_2 ("Please wait...") created 1109597298.469

CtxWaitForWindowCreate(CWI_2, 31);

DO_MSLEEP(47);
CtxPoint(324, 353);  //1109597298.500

// Window CWI_2 ("Please wait...") destroyed 1109597298.500

CtxPoint(321, 325);  //1109597298.578

// Window CWI_3 ("Citrix License Warning Notice") created 1109597298.578

CtxWaitForWindowCreate(CWI_3, 15);

// Window CWI_4 ("UsrLogon.Cmd") created 1109597298.578

CtxWaitForWindowCreate(CWI_4, 0);

DO_MSLEEP(16);
// Window CWI_4 ("UsrLogon.Cmd") destroyed 1109597298.594

CtxPoint(307, 268);  //1109597298.766

// Window CWI_5 ("") created 1109597298.766

DO_MSLEEP(63);
// Window CWI_6 ("ICA Seamless Host Agent") created 1109597298.828

CtxWaitForWindowCreate(CWI_6, 62);

// Window CWI_7 ("Program Manager") created 1109597298.828

CtxWaitForWindowCreate(CWI_7, 0);

// Window CWI_8 ("") created 1109597301.123
```

```
DO_MSLEEP(2297);

DO_MSLEEP(2797);
// Window CWI_8 ("") destroyed 1109597303.918

DO_MSLEEP(5438);
CtxPoint(289, 212);  //1109597309.524

DO_MSLEEP(171);
CtxClick(CWI_3, 266, L_BUTTON, NONE); //1109597309.789

DO_MSLEEP(1109);
CtxPoint(329, 295);  //1109597310.944

DO_MSLEEP(47);
CtxClick(CWI_3, 250, L_BUTTON, NONE); //1109597311.194

DO_MSLEEP(63);
// Window CWI_3 ("Citrix License Warning Notice") destroyed 1109597311.257

DO_MSLEEP(984);
CtxPoint(261, 132);  //1109597312.475

DO_MSLEEP(235);
CtxClick(CWI_6, 312, L_BUTTON, NONE); //1109597312.787

DO_MSLEEP(563);
CtxPoint(208, 194);  //1109597313.708

DO_MSLEEP(359);
CtxClick(CWI_6, 281, L_BUTTON, NONE); //1109597313.989

DO_MSLEEP(78);
// Window CWI_6 ("ICA Seamless Host Agent") destroyed 1109597314.067

/*
After Logon
*/
DO_MSLEEP(29578);
CtxPoint(101, 96);  //1109597343.983

DO_MSLEEP(360);
CtxDoubleClick(CWI_7); // 1109597344.186

CtxMouseUp(CWI_5, L_BUTTON, NONE, 100, 96); //1109597344.093

DO_MSLEEP(47);
CtxPoint(100, 96);  //1109597344.186

DO_MSLEEP(156);

// Window CWI_9 ("Calculator") created 1109597344.327

CtxWaitForWindowCreate(CWI_9, 141);

DO_MSLEEP(921);
CtxPoint(181, 195);  //1109597345.420

DO_MSLEEP(172);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 181, 195);  // 1109597345.420

CtxMouseUp(CWI_9, L_BUTTON, NONE, 181, 196); //1109597345.529

DO_MSLEEP(438);
```

```
CtxPoint(210, 196);  //1109597345.857

CtxMouseDown(CWI_9, L_BUTTON, NONE, 210, 196);  // 1109597345.857

CtxMouseUp(CWI_9, L_BUTTON, NONE, 211, 197); //1109597346.013

DO_MSLEEP(562);
CtxPoint(252, 230);  //1109597346.950

DO_MSLEEP(532);
CtxClick(CWI_9, 140, L_BUTTON, NONE); //1109597347.091

DO_MSLEEP(531);
CtxPoint(171, 222);  //1109597347.653

DO_MSLEEP(32);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 171, 222);  // 1109597347.653

CtxMouseUp(CWI_9, L_BUTTON, NONE, 172, 222); //1109597347.746

DO_MSLEEP(547);
CtxPoint(231, 231);  //1109597348.215

DO_MSLEEP(15);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 231, 231);  // 1109597348.215

CtxMouseUp(CWI_9, L_BUTTON, NONE, 230, 231); //1109597348.340

DO_MSLEEP(953);
CtxPoint(303, 293);  //1109597349.370

DO_MSLEEP(188);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 303, 293);  // 1109597349.370

CtxMouseUp(CWI_9, L_BUTTON, NONE, 302, 293); //1109597349.480

DO_MSLEEP(1859);
CtxPoint(314, 79);  //1109597351.494

DO_MSLEEP(281);
CtxClick(CWI_9, 157, L_BUTTON, NONE); //1109597351.650

DO_MSLEEP(62);
// Window CWI_9 ("Calculator") destroyed 1109597351.712

/*
Before Logoff
*/
DO_MSLEEP(33781);
CtxPoint(384, 13);  //1109597396.040

DO_MSLEEP(10579);
// Window CWI_7 ("Program Manager") destroyed 1109597396.040
// Window CWI_5 ("") destroyed 1109597396.040

DO_SetTransactionCleanup();
CtxDisconnect();

END_TRANSACTION();
delete CWI_1; // "Warning !!"
delete CWI_2; // "Please wait..."
delete CWI_3; // "Citrix License Warning Notice"
delete CWI_4; // "UsrLogon.Cmd"
delete CWI_5; // ""
delete CWI_6; // "ICA Seamless Host Agent"
```

```
    delete CWI_7; // "Program Manager"
    delete CWI_8; // ""
    delete CWI_9; // "Calculator"

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Sample: Modified Citrix Script for Modifying the Transaction Loop

The following sample is a modified Citrix script that moves the transaction loop. Changes to the original script are shown in bold.

Sample Script

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Citrix
 *  General Options                  :
 *   Replay Output Mode              : Normal
 *   Enable Counters                 : No
 *  Timeout Value Options            :
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */
#define CITRIX_CLIENT_VERSION "8.100.29670"
#define CITRIX_ICO_VERSION    "2.4"
#define SCRIPT_VER 0x00000505UL
#include <stdio.h>
#include "smacro.h"
#include "do_citrix.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
    /* Declare Variables */
```

```
    const char *CitrixServer      = "QACCitrix";
    const char *CitrixUsername    = "Citrix";
    const char *CitrixPassword    = "~encr~657E06726F697206";
    const char *CitrixDomain      = "QACCitrix";
    const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

    /* Citrix Window Information Objects */
    CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
    CtxWI *CWI_2 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
    CtxWI *CWI_3 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
    CtxWI *CWI_4 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
    CtxWI *CWI_5 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
    CtxWI *CWI_6 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
    CtxWI *CWI_7 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
    CtxWI *CWI_8 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
    CtxWI *CWI_9 = new CtxWI(0x2006c, "Calculator", 66, 66, 261, 253);

    SET_ABORT_FUNCTION(abort_function);

    DEFINE_TRANS_TYPE("capCtxMoveLoop.cpp");

    CitrixInit(1);

    /* Citrix replay settings */
    CtxSetConnectTimeout(60);
    CtxSetDisconnectTimeout(60);
    CtxSetWindowTimeout(30);
    CtxSetPingTimeout(20);
    CtxSetWaitPointTimeout(30);
    CtxSetWindowVerification(TRUE);
    CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, CitrixDomain);
    CtxSetEnableCounters(FALSE);
    CtxSetWindowRetries(5, 5000);
    CtxSetEnableWildcardMatching(TRUE);

    SYNCHRONIZE();
    /////////////////////////////////////////////
    // Remove the BEGIN_TRANSACTION();
    // and place it after the logon comment
    // in the script
    /////////////////////////////////////////////

// BEGIN_TRANSACTION();

    /////////////////////////////////////////////
    // Remove the DO_SetTransactionStart();
    // Restart transaction cannot be used if the
    // Transaction Loop has been moved
    /////////////////////////////////////////////

// DO_SetTransactionStart();

    CtxConnect(CitrixServer, CitrixOutputMode);

    // Window CWI_1 ("Warning !!") created 1109597295.502

    CtxWaitForWindowCreate(CWI_1, 2156);

    DO_MSLEEP(922);
    CtxPoint(333, 233);  //1109597296.611

    DO_MSLEEP(188);
    CtxMouseDown(CWI_1, L_BUTTON, NONE, 333, 233);  // 1109597296.611

    CtxMouseUp(CWI_1, L_BUTTON, NONE, 332, 233); //1109597296.861
```

```
DO_MSLEEP(1250);
CtxPoint(319, 360);  //1109597297.938

DO_MSLEEP(78);
CtxMouseDown(CWI_1, L_BUTTON, NONE, 319, 360);  // 1109597297.938

CtxMouseUp(CWI_1, L_BUTTON, NONE, 320, 360); //1109597298.235

DO_MSLEEP(78);
CtxPoint(320, 360);  //1109597298.282

DO_MSLEEP(265);
// Window CWI_1 ("Warning !!") destroyed 1109597298.282

CtxPoint(321, 358);  //1109597298.469

// Window CWI_2 ("Please wait...") created 1109597298.469

CtxWaitForWindowCreate(CWI_2, 31);

DO_MSLEEP(47);
CtxPoint(324, 353);  //1109597298.500

// Window CWI_2 ("Please wait...") destroyed 1109597298.500

CtxPoint(321, 325);  //1109597298.578

// Window CWI_3 ("Citrix License Warning Notice") created 1109597298.578

CtxWaitForWindowCreate(CWI_3, 15);

// Window CWI_4 ("UsrLogon.Cmd") created 1109597298.578

CtxWaitForWindowCreate(CWI_4, 0);

DO_MSLEEP(16);
// Window CWI_4 ("UsrLogon.Cmd") destroyed 1109597298.594

CtxPoint(307, 268);  //1109597298.766

// Window CWI_5 ("") created 1109597298.766

DO_MSLEEP(63);
// Window CWI_6 ("ICA Seamless Host Agent") created 1109597298.828

CtxWaitForWindowCreate(CWI_6, 62);

// Window CWI_7 ("Program Manager") created 1109597298.828

CtxWaitForWindowCreate(CWI_7, 0);

// Window CWI_8 ("") created 1109597301.123

DO_MSLEEP(2297);

DO_MSLEEP(2797);
// Window CWI_8 ("") destroyed 1109597303.918

DO_MSLEEP(5438);
CtxPoint(289, 212);  //1109597309.524

DO_MSLEEP(171);
CtxClick(CWI_3, 266, L_BUTTON, NONE); //1109597309.789
```

```
DO_MSLEEP(1109);
CtxPoint(329, 295);  //1109597310.944

DO_MSLEEP(47);
CtxClick(CWI_3, 250, L_BUTTON, NONE); //1109597311.194

DO_MSLEEP(63);
// Window CWI_3 ("Citrix License Warning Notice") destroyed 1109597311.257

DO_MSLEEP(984);
CtxPoint(261, 132);  //1109597312.475

DO_MSLEEP(235);
CtxClick(CWI_6, 312, L_BUTTON, NONE); //1109597312.787

DO_MSLEEP(563);
CtxPoint(208, 194);  //1109597313.708

DO_MSLEEP(359);
CtxClick(CWI_6, 281, L_BUTTON, NONE); //1109597313.989

DO_MSLEEP(78);
// Window CWI_6 ("ICA Seamless Host Agent") destroyed 1109597314.067

/*
After Logon
*/
/////////////////////////////////////////////
// Move Begin Transaction Loop to after
// the Logon comment created during capture
// process You may want to reduce the sleep
// time introduced into the process by adding
// the comment
/////////////////////////////////////////////
BEGIN_TRANSACTION();

DO_MSLEEP(2578);
CtxPoint(101, 96);  //1109597343.983

DO_MSLEEP(360);
CtxDoubleClick(CWI_7); // 1109597344.186

CtxMouseUp(CWI_5, L_BUTTON, NONE, 100, 96); //1109597344.093

DO_MSLEEP(47);
CtxPoint(100, 96);  //1109597344.186

DO_MSLEEP(156);

// Window CWI_9 ("Calculator") created 1109597344.327

CtxWaitForWindowCreate(CWI_9, 141);

DO_MSLEEP(921);
CtxPoint(181, 195);  //1109597345.420

DO_MSLEEP(172);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 181, 195);  // 1109597345.420

CtxMouseUp(CWI_9, L_BUTTON, NONE, 181, 196); //1109597345.529

DO_MSLEEP(438);
CtxPoint(210, 196);  //1109597345.857

CtxMouseDown(CWI_9, L_BUTTON, NONE, 210, 196);  // 1109597345.857
```

```
CtxMouseUp(CWI_9, L_BUTTON, NONE, 211, 197); //1109597346.013

DO_MSLEEP(562);
CtxPoint(252, 230);  //1109597346.950

DO_MSLEEP(532);
CtxClick(CWI_9, 140, L_BUTTON, NONE); //1109597347.091

DO_MSLEEP(531);
CtxPoint(171, 222);  //1109597347.653

DO_MSLEEP(32);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 171, 222);  // 1109597347.653

CtxMouseUp(CWI_9, L_BUTTON, NONE, 172, 222); //1109597347.746

DO_MSLEEP(547);
CtxPoint(231, 231);  //1109597348.215

DO_MSLEEP(15);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 231, 231);  // 1109597348.215

CtxMouseUp(CWI_9, L_BUTTON, NONE, 230, 231); //1109597348.340

DO_MSLEEP(953);
CtxPoint(303, 293);  //1109597349.370

DO_MSLEEP(188);
CtxMouseDown(CWI_9, L_BUTTON, NONE, 303, 293);  // 1109597349.370

CtxMouseUp(CWI_9, L_BUTTON, NONE, 302, 293); //1109597349.480

DO_MSLEEP(1859);
CtxPoint(314, 79);  //1109597351.494

DO_MSLEEP(281);
CtxClick(CWI_9, 157, L_BUTTON, NONE); //1109597351.650

DO_MSLEEP(62);
// Window CWI_9 ("Calculator") destroyed 1109597351.712

/*
Before Logoff
*/
/////////////////////////////////////////////
// Move End Transaction Loop to before
// the Logoff comment created during capture
// process You may want to reduce the sleep
// time introduced into the process by adding
// the comment
/////////////////////////////////////////////
END_TRANSACTION();

DO_MSLEEP(3781);
CtxPoint(384, 13);  //1109597396.040

DO_MSLEEP(1079);
// Window CWI_7 ("Program Manager") destroyed 1109597396.040

// Window CWI_5 ("") destroyed 1109597396.040
/////////////////////////////////////////////
// Remove the DO_SetTransactionCleanup();
// Restart transaction cannot be used if the
// Transaction Loop has been moved
```

```
    ///////////////////////////////////////////////

// DO_SetTransactionCleanup();

    CtxDisconnect();

    ///////////////////////////////////////////////
    // Remove the END_TRANSACTION();
    // and place it before the logoff comment
    // in the script
    ///////////////////////////////////////////////

// END_TRANSACTION();
    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Please wait..."
    delete CWI_3; // "Citrix License Warning Notice"
    delete CWI_4; // "UsrLogon.Cmd"
    delete CWI_5; // ""
    delete CWI_6; // "ICA Seamless Host Agent"
    delete CWI_7; // "Program Manager"
    delete CWI_8; // ""
    delete CWI_9; // "Calculator"

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Scripting Mouse Actions

Modifying the Script for Controlling Mouse Actions

Changes to screen resolution or server settings can cause the appearance, size, and location coordinates of a window to differ from the expected behavior in the recorded script. By setting Display coordinate tooltip on right-click option in validation window in the Convert Options dialog box, you can right-click the mouse at the desired location to extract screen coordinate values that correspond to buttons and other controls in real time during validation .

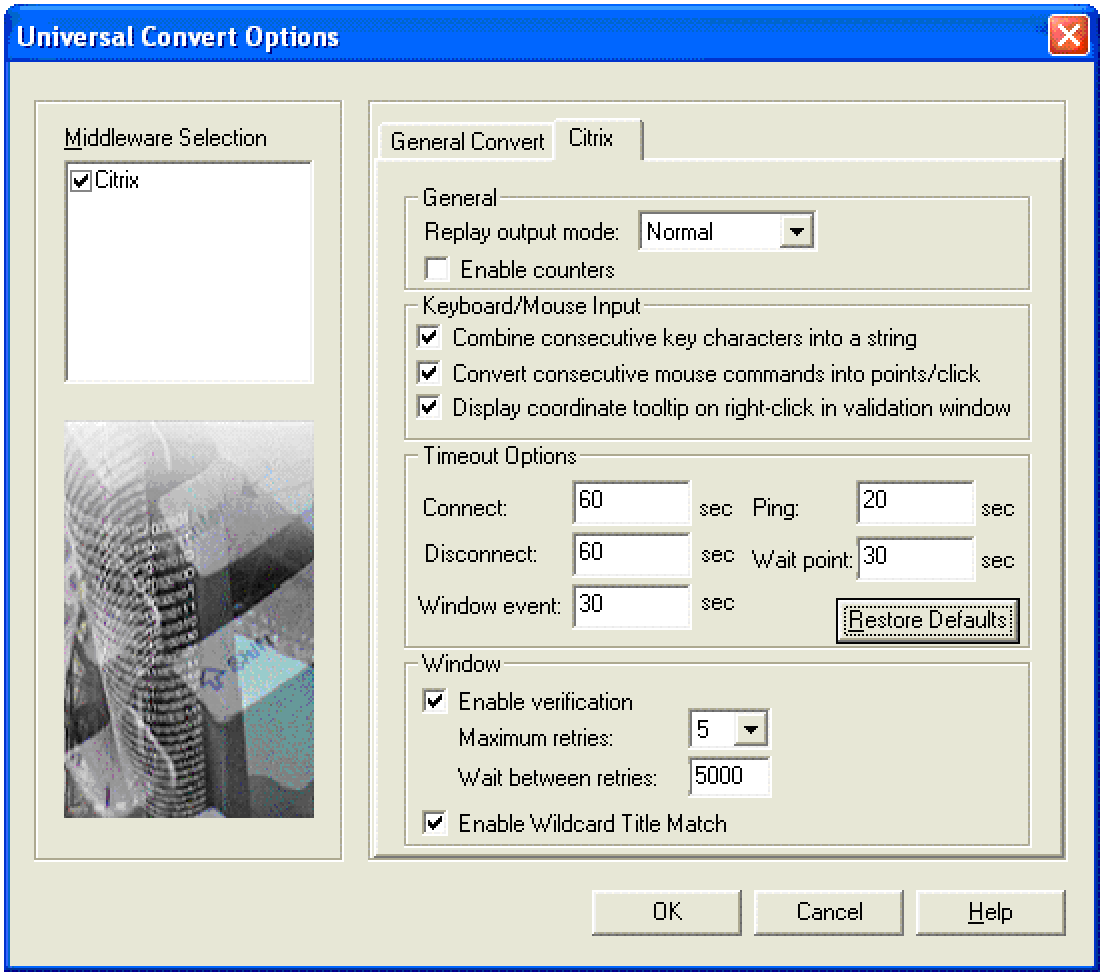
To modify a script to retrieve coordinate values, you must:

- Validate the settings.
- Modify the script.

To validate the settings:

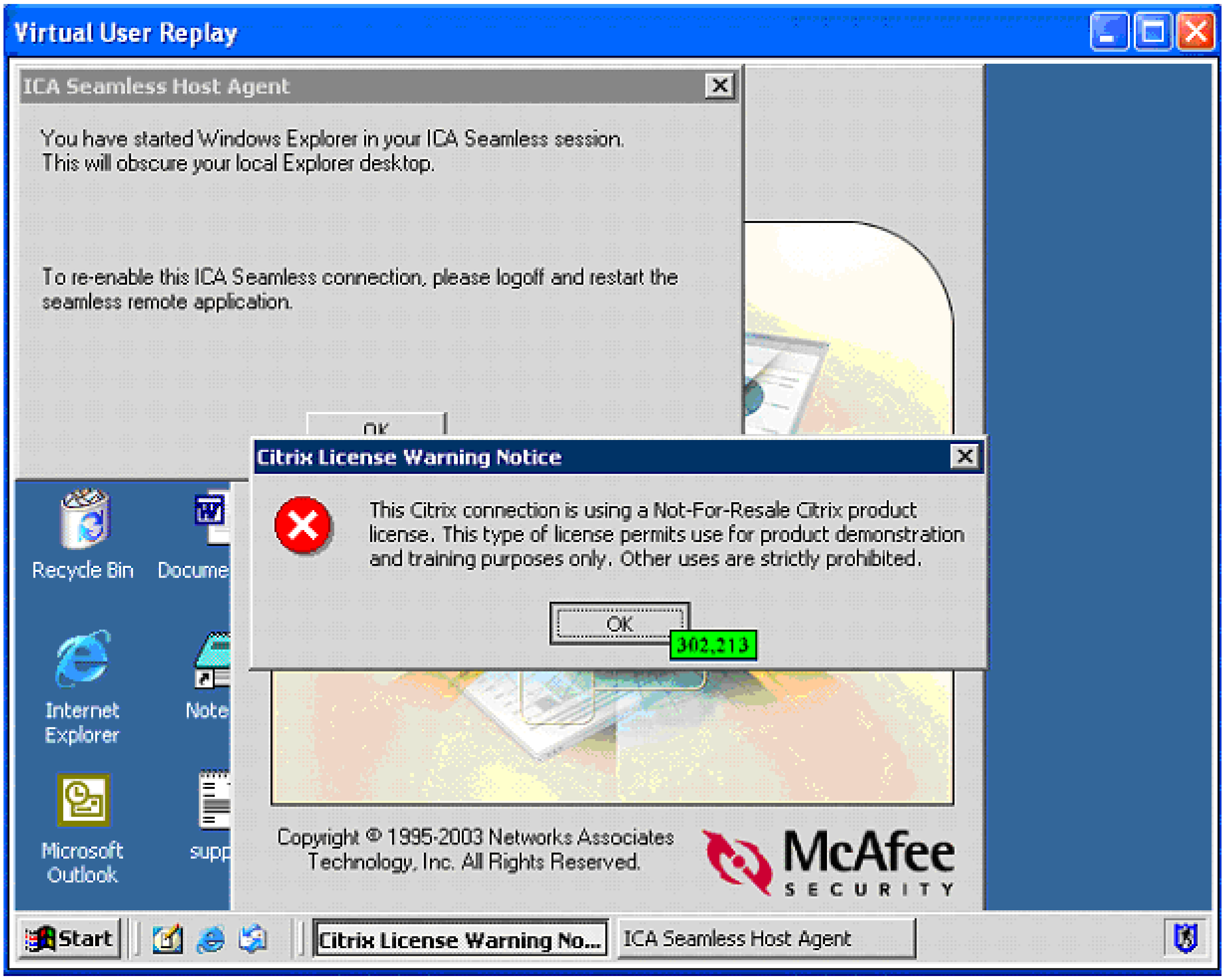
1. Click Options>Convert, and click the Citrix tab.
2. Ensure that the Display coordinate tooltip on right-click option in validation window checkbox is selected.
3. Set Replay output mode to Normal.

4. Click OK to convert the script.



During validation, a right mouse-click results in a tooltip display of the screen coordinate values. You can use these values in a manually added CtxPoint call to ensure that the mouse is moved to these coordinates before any subsequent mouse click actions.

Tip: Make a note of these coordinate values, since this tooltip is not logged.



To modify the script:

Once you determine the correct mouse coordinates from validation, modify the script.

1. Add or modify the CtxPoint command.
2. Insert the new values into the script.

Conclusion

Following these techniques, you can retrieve screen coordinates at validation and insert them into a script. This corrects mouse behavior as a result of changes to the server settings or the user session environment. The sample scripts, showing added CtxPoint and CtxMouseClick, and modified CtxPoint, illustrate these techniques.

Sample: Script Snippet with Added CtxPoint and CtxClick

The following sample shows a script extract with CtxPoint and CtxClick commands added. Comments and added script lines are highlighted in bold.

Sample Script Snippet

```
…beginning of script…
   DO_MSLEEP(687);
```

```
/// This CtxPoint is for an intermittent window
/// that appeared. A button is required to dismiss
/// this window. Right-clicking on the button during
/// validation gave the coordinates in a tooltip.

   CtxPoint(268, 60);

/// A DO_MSLEEP will pause execution for a second.
   DO_MSLEEP(1000);
/// This CtxSetWindowMatchName is for the intermittent window name.

   SetWindowMatchName( CWI_9, "*" );

/// This CtxClick command is added to dismiss the window.
   CtxClick(CWI_9, 78, L_BUTTON, NONE);
…end of script…
```

Sample: Script Snippet with Modified CtxPoint

The following sample shows a script extract with a modified CtxPoint command. Comments and added script lines are highlighted in bold.

Sample Script

```
…beginning of script…

   DO_MSLEEP(1844);
   // Window CWI_10 ("") destroyed 1113840796.133

   DO_MSLEEP(1828);

/// Modifying a CtxPoint is easy once you have the
/// target coordinates and have identified the
/// CtxClick statement that was failing. Just
/// insert the X and Y coordinates as the parameters.

   CtxPoint(200, 200);  //1113840798.309

   DO_MSLEEP(344);
   CtxClick(CWI_5, 78, L_BUTTON, NONE); //1113840798.387

…end of script…
```

Using CtxWaitForScreenUpdate Command

Modifying the Script with the CtxWaitForScreenUpdate Command

Windows in a Citrix session can take varying amounts of time to complete screen updates. This can result in keyboard and mouse events not synchronizing with the appropriate window.

For example, the Windows Start menu is an unnamed window. The amount of time it takes to appear depends on system resource usage. Adding a CtxWaitForScreenUpdate command before the keyboard entry or mouse click can ensure that the window that the window has enough time to display before any subsequent mouse clicks or keyboard entries take place. You can add the command while recording a script or after a script has been recorded.

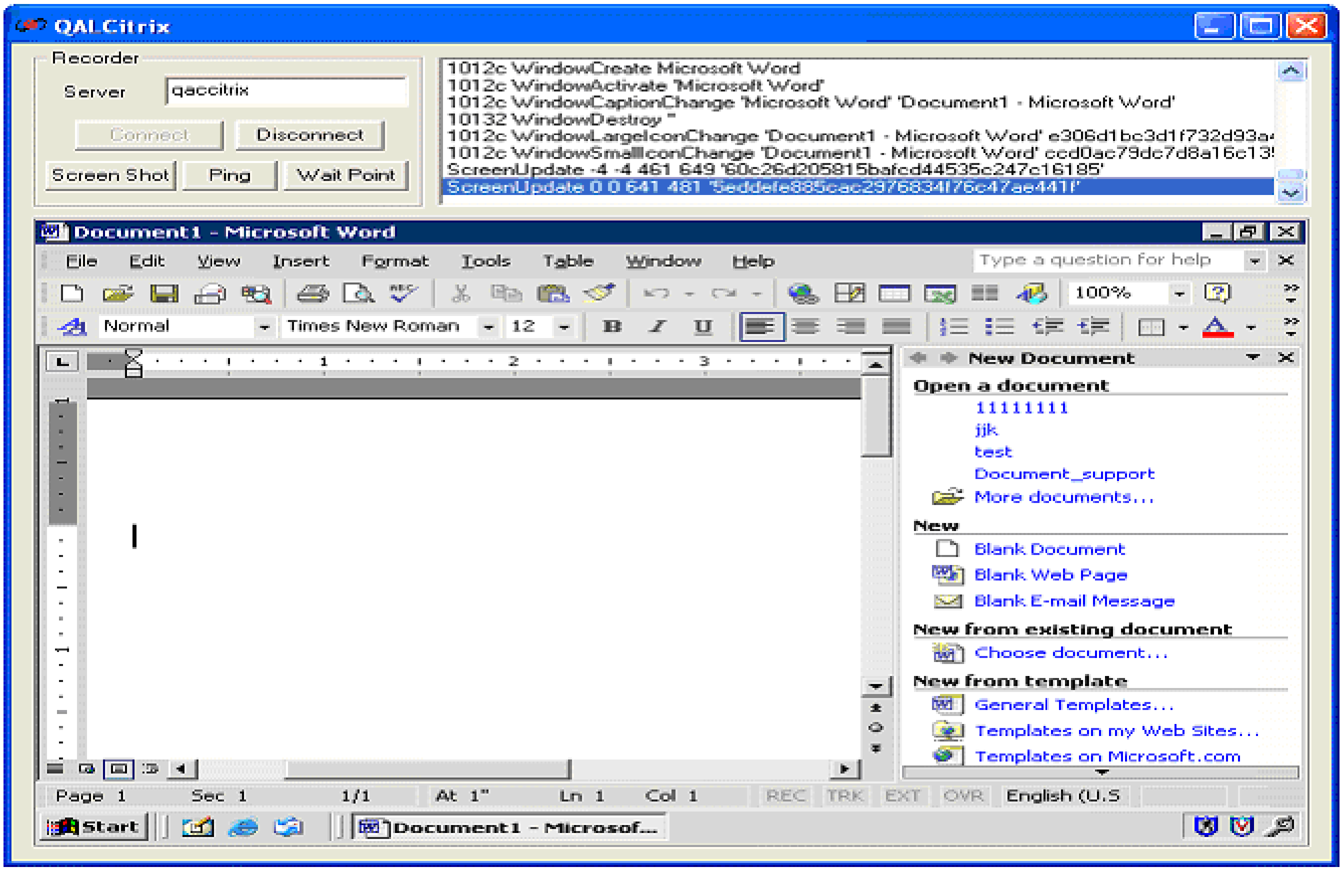
> Note: When recording a session with windows that load slowly, pay attention to the ScreenUpdate messages that scroll in the message panel.

Once the window has completed loading, do the following:

To add CtxWaitForScreenUpdate while recording a session:

1. Click the last ScreenUpdate message to highlight the line.
2. Click WaitPoint to insert a statement in the record session. The statement is converted to a CtxWaitForScreenUpdate command in the script, which waits for the screen update before proceeding.

Note: Anytime a window loads slowly or has the potential to load slowly during a Citrix record session, such as when traffic is heavy, consider using this technique before performing any keyboard or mouse activity on that window.



To add CtxWaitForScreenUpdate to an existing script:

Manually enter the CtxWaitForScreenUpdate before the keyboard or mouse event that is firing prematurely.

Note: To prevent playback problems in which a mouse click does not synchronize with its intended window, insert the command in the script after the action that causes the window to appear.

The parameters for the CtxWaitForScreenUpdate command correspond to the X and Y coordinates, and the width and height of the window. You can determine the parameters from the Citrix window object declaration parameters.

Conclusion

Following these techniques, you can prevent problems with premature mouse clicks and keyboard entry statements during replay. The sample scripts, for using the command while recording and for using the command after recording, illustrate this technique.

Sample: Using the Command After Recording

The following sample shows a script with WaitPoint inserted after recording is completed. Script changes are highlighted in bold.

Sample Script

```
/*
 *  waitforscreenupdate.cpp
 *
 *  Script Converted on April 19, 2005 at 02:44:44 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Citrix
 *  General Options                  :
 *   Replay Output Mode              : Normal
 *   Enable Counters                 : No
 *  Timeout Value Options            :
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */
#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   const char *CitrixServer      = "qaccitrix";
   const char *CitrixPassword    = "";
   const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;
```

```
/* Citrix Window Information Objects */
CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
CtxWI *CWI_4 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
CtxWI *CWI_5 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
CtxWI *CWI_6 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
CtxWI *CWI_7 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
CtxWI *CWI_8 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
CtxWI *CWI_9 = new CtxWI(0x10088, "", 115, 0, 405, 457);
CtxWI *CWI_10 = new CtxWI(0x10134, "", 112, 116, 416, 248);
CtxWI *CWI_11 = new CtxWI(0x1012e, "Microsoft Word", 0, 0, 649, 461);
CtxWI *CWI_12 = new CtxWI(0x10176, "Microsoft Word", 162, 193, 322, 109);
CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("waitforscreenupdate.cpp");

CitrixInit(1);

/* Citrix replay settings */
CtxSetConnectTimeout(60);
CtxSetDisconnectTimeout(60);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetEnableCounters(FALSE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

CtxPoint(202, 89);  //1113936216.354

// Window CWI_1 ("Warning !!") created 1113936216.354

CtxWaitForWindowCreate(CWI_1, 1953);

DO_MSLEEP(1859);
CtxPoint(320, 372);  //1113936218.260

DO_MSLEEP(47);
CtxClick(CWI_1, 78, L_BUTTON, NONE); //1113936218.338

// Window CWI_2 ("Log On to Windows") created 1113936218.479

CtxWaitForWindowCreate(CWI_2, 141);

// Window CWI_1 ("Warning !!") destroyed 1113936218.479

CtxType(CWI_2, "citrix");  //1113936220.995

DO_MSLEEP(2312);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113936221.073

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113936223.104
```

```
DO_MSLEEP(1782);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113936223.151

DO_MSLEEP(297);
CtxType(CWI_2, "q");  //1113936224.354

DO_MSLEEP(718);

DO_MSLEEP(1407);
CtxPoint(243, 326);  //1113936225.291

DO_MSLEEP(15);
CtxClick(CWI_2, 63, L_BUTTON, NONE); //1113936225.354

DO_MSLEEP(78);
// Window CWI_2 ("Log On to Windows") destroyed 1113936225.432

// Window CWI_3 ("Please wait...") created 1113936225.510

CtxWaitForWindowCreate(CWI_3, 78);

DO_MSLEEP(78);
// Window CWI_3 ("Please wait...") destroyed 1113936225.588

// Window CWI_4 ("Citrix License Warning Notice") created 1113936225.651

CtxWaitForWindowCreate(CWI_4, 63);

// Window CWI_5 ("UsrLogon.Cmd") created 1113936225.666

CtxWaitForWindowCreate(CWI_5, 15);

DO_MSLEEP(16);
// Window CWI_5 ("UsrLogon.Cmd") destroyed 1113936225.682

// Window CWI_6 ("") created 1113936225.916

DO_MSLEEP(234);
// Window CWI_7 ("ICA Seamless Host Agent") created 1113936225.963

CtxWaitForWindowCreate(CWI_7, 47);

// Window CWI_8 ("Program Manager") created 1113936225.979

CtxWaitForWindowCreate(CWI_8, 16);

DO_MSLEEP(953);
CtxPoint(332, 299);  //1113936226.963

DO_MSLEEP(31);
CtxClick(CWI_7, 78, L_BUTTON, NONE); //1113936227.041

DO_MSLEEP(63);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113936227.104

DO_MSLEEP(328);
CtxPoint(198, 203);  //1113936227.588

DO_MSLEEP(156);
CtxClick(CWI_7, 63, L_BUTTON, NONE); //1113936227.651

DO_MSLEEP(78);
CtxPoint(208, 216);  //1113936227.744
```

```
    DO_MSLEEP(15);
    // Window CWI_7 ("ICA Seamless Host Agent") destroyed 1113936227.744

    CtxPoint(266, 237);  //1113936228.260

    // Window CWI_9 ("") created 1113936228.260

    DO_MSLEEP(453);

    DO_MSLEEP(3031);
    // Window CWI_9 ("") destroyed 1113936231.291

    DO_MSLEEP(8531);
    CtxPoint(115, 170);  //1113936240.182

    DO_MSLEEP(360);
    CtxDoubleClick(CWI_8); // 1113936240.338

    DO_MSLEEP(78);

    // Window CWI_10 ("") created 1113936240.463

    DO_MSLEEP(125);
    CtxPoint(243, 205);  //1113936240.619

    // Window CWI_11 ("Microsoft Word") created 1113936240.619

    CtxWaitForWindowCreate(CWI_11, 156);

    // Window CWI_10 ("") destroyed 1113936240.619

    DO_MSLEEP(32);
    CtxPoint(250, 208);  //1113936240.666

    DO_MSLEEP(15);
    CWI_11->setTitle("Document1 - Microsoft Word"); //1113936240.869

    CtxPoint(296, 0);  //1113936245.916

/// This command was added to the script manually.  The size parameters
/// are from the Citrix window creation object, and the last parameter
/// indicates that the wait time for completion of this screen
/// update is set to 5 seconds.  Note that this statement is
/// placed before any mouse moves for that window.

    CtxWaitForScreenUpdate(0, 0, 649, 481, 5000);


    DO_MSLEEP(2094);
    CtxPoint(56, 201);  //1113936248.260

    DO_MSLEEP(250);
    CtxClick(CWI_11, 47, L_BUTTON, NONE); //1113936248.307

    CtxType(CWI_11, "N");  //1113936249.760

    DO_MSLEEP(1453);
    CtxPoint(68, 213);  //1113936249.760

    CtxType(CWI_11, "ow I");  //1113936251.041

    DO_MSLEEP(1281);
    CtxType(CWI_11, " can type text!");  //1113936255.119

    DO_MSLEEP(4078);
```

```
    DO_MSLEEP(2766);

    DO_MSLEEP(2094);
    CtxPoint(635, 5);  //1113936260.010

    DO_MSLEEP(31);
    CtxClick(CWI_11, 78, L_BUTTON, NONE); //1113936260.088

    CtxPoint(634, 12);  //1113936260.213

    // Window CWI_12 ("Microsoft Word") created 1113936260.213

    CtxWaitForWindowCreate(CWI_12, 16);

    DO_MSLEEP(875);
    CtxPoint(325, 275);  //1113936261.088

    CtxClick(CWI_12, 63, L_BUTTON, NONE); //1113936261.151

    DO_MSLEEP(62);
    // Window CWI_12 ("Microsoft Word") destroyed 1113936261.213

    DO_MSLEEP(16);
    // Window CWI_11 ("Document1 - Microsoft Word") destroyed 1113936261.229

    DO_MSLEEP(1375);
    CtxPoint(385, 165);  //1113936262.963

    DO_MSLEEP(359);
    // Window CWI_8 ("Program Manager") destroyed 1113936262.963

    // Window CWI_6 ("") destroyed 1113936262.979

    // Window CWI_13 ("Please wait...") created 1113936263.244

    CtxWaitForWindowCreate(CWI_13, 0);

    DO_SetTransactionCleanup();

    CtxDisconnect();

    END_TRANSACTION();

    delete CWI_1; // "Warning !!"
    delete CWI_2; // "Log On to Windows"
    delete CWI_3; // "Please wait..."
    delete CWI_4; // "Citrix License Warning Notice"
    delete CWI_5; // "UsrLogon.Cmd"
    delete CWI_6; // ""
    delete CWI_7; // "ICA Seamless Host Agent"
    delete CWI_8; // "Program Manager"
    delete CWI_9; // ""
    delete CWI_10; // ""
    delete CWI_11; // "Microsoft Word"
    delete CWI_12; // "Microsoft Word"
    delete CWI_13; // "Please wait..."

    CitrixUninit();

    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
```

```
    RR__printf("Virtual User ABORTED.");

    CitrixUninit();

    EXIT();
}
```

Sample: Using the Command While Recording

The following sample shows a script with the WaitPoint command inserted at record time. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  waitforscreenupdate.cpp
 *
 *  Script Converted on April 19, 2005 at 02:44:44 PM
 *  Generated by Compuware QALoad convert module version 5.5.0 build 256
 *
 *  This script contains support for the following middlewares:
 *      - Citrix
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Citrix
 *  General Options                  :
 *   Replay Output Mode              : Normal
 *   Enable Counters                 : No
 *  Timeout Value Options            :
 *    Connect Timeout (s)            : 60
 *    Disconnect Timeout (s)         : 60
 *    Window Creation Timeout (s)    : 30
 *    Ping Timeout (s)               : 20
 *    Wait Point Timeout (s)         : 30
 *  Input Options                    :
 *   Combine Keyboard Input          : Yes
 *   Combine Mouse Input             : Yes
 *  Window Options                   :
 *   Window Verification             : Yes
 *    Window Max Retries             : 5
 *    Window Wait Retries (ms)       : 5000
 *   Enable Wildcard Title Match     : Yes
 */

#define CITRIX_CLIENT_VERSION "7.100.21825"
#define CITRIX_ICO_VERSION    "2.3"
#define SCRIPT_VER 0x00000505UL

#include <stdio.h>
#include "smacro.h"

#include "do_citrix.h"

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif
```

```
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
     /* Declare Variables */
     const char *CitrixServer      = "qaccitrix";
     const char *CitrixPassword    = "";
     const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

     /* Citrix Window Information Objects */
     CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
     CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
     CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
     CtxWI *CWI_4 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198, 397, 127);
     CtxWI *CWI_5 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
     CtxWI *CWI_6 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
     CtxWI *CWI_7 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
     CtxWI *CWI_8 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
     CtxWI *CWI_9 = new CtxWI(0x10088, "", 115, 0, 405, 457);
     CtxWI *CWI_10 = new CtxWI(0x10134, "", 112, 116, 416, 248);
     CtxWI *CWI_11 = new CtxWI(0x1012e, "Microsoft Word", -4, -4, 649, 461);
     CtxWI *CWI_12 = new CtxWI(0x10176, "Microsoft Word", 162, 193, 322, 109);
     CtxWI *CWI_13 = new CtxWI(0x7001c, "Please wait...", 111, 112, 418, 145);

     SET_ABORT_FUNCTION(abort_function);

     DEFINE_TRANS_TYPE("waitforscreenupdate.cpp");

     CitrixInit(1);

     /* Citrix replay settings */
     CtxSetConnectTimeout(60);
     CtxSetDisconnectTimeout(60);
     CtxSetWindowTimeout(30);
     CtxSetPingTimeout(20);
     CtxSetWaitPointTimeout(30);
     CtxSetWindowVerification(TRUE);
     CtxSetEnableCounters(FALSE);
     CtxSetWindowRetries(5, 5000);
     CtxSetEnableWildcardMatching(TRUE);

     SYNCHRONIZE();

     BEGIN_TRANSACTION();

     DO_SetTransactionStart();

     CtxConnect(CitrixServer, CitrixOutputMode);

     CtxPoint(202, 89);  //1113936216.354

     // Window CWI_1 ("Warning !!") created 1113936216.354

     CtxWaitForWindowCreate(CWI_1, 1953);

     DO_MSLEEP(1859);
     CtxPoint(320, 372);  //1113936218.260

     DO_MSLEEP(47);
     CtxClick(CWI_1, 78, L_BUTTON, NONE); //1113936218.338

     // Window CWI_2 ("Log On to Windows") created 1113936218.479

     CtxWaitForWindowCreate(CWI_2, 141);

     // Window CWI_1 ("Warning !!") destroyed 1113936218.479
```

```
CtxType(CWI_2, "citrix");  //1113936220.995

DO_MSLEEP(2312);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113936221.073

DO_MSLEEP(281);
CtxType(CWI_2, "citrix");  //1113936223.104

DO_MSLEEP(1782);
CtxTypeVK(CWI_2, VK_TAB, NONE);  //1113936223.151

DO_MSLEEP(297);
CtxType(CWI_2, "q");  //1113936224.354

DO_MSLEEP(718);

DO_MSLEEP(1407);
CtxPoint(243, 326);  //1113936225.291

DO_MSLEEP(15);
CtxClick(CWI_2, 63, L_BUTTON, NONE); //1113936225.354

DO_MSLEEP(78);
// Window CWI_2 ("Log On to Windows") destroyed 1113936225.432

// Window CWI_3 ("Please wait...") created 1113936225.510

CtxWaitForWindowCreate(CWI_3, 78);

DO_MSLEEP(78);
// Window CWI_3 ("Please wait...") destroyed 1113936225.588

// Window CWI_4 ("Citrix License Warning Notice") created

CtxWaitForWindowCreate(CWI_4, 63);

// Window CWI_5 ("UsrLogon.Cmd") created 1113936225.666

CtxWaitForWindowCreate(CWI_5, 15);

DO_MSLEEP(16);
// Window CWI_5 ("UsrLogon.Cmd") destroyed 1113936225.682

// Window CWI_6 ("") created 1113936225.916

DO_MSLEEP(234);
// Window CWI_7 ("ICA Seamless Host Agent") created 1113936225.963

CtxWaitForWindowCreate(CWI_7, 47);

// Window CWI_8 ("Program Manager") created 1113936225.979

CtxWaitForWindowCreate(CWI_8, 16);

DO_MSLEEP(953);
CtxPoint(332, 299);  //1113936226.963

DO_MSLEEP(31);
CtxClick(CWI_7, 78, L_BUTTON, NONE); //1113936227.041

DO_MSLEEP(63);
// Window CWI_4 ("Citrix License Warning Notice") destroyed 1113936227.104

DO_MSLEEP(328);
CtxPoint(198, 203);  //1113936227.588
```

```
DO_MSLEEP(156);
CtxClick(CWI_7, 63, L_BUTTON, NONE); //1113936227.651

DO_MSLEEP(78);
CtxPoint(208, 216);  //1113936227.744

DO_MSLEEP(15);
// Window CWI_7 ("ICA Seamless Host Agent") destroyed 1113936227.744

CtxPoint(266, 237);  //1113936228.260

// Window CWI_9 ("") created 1113936228.260

DO_MSLEEP(453);

DO_MSLEEP(3031);
// Window CWI_9 ("") destroyed 1113936231.291

DO_MSLEEP(8531);
CtxPoint(115, 170);  //1113936240.182

DO_MSLEEP(360);
CtxDoubleClick(CWI_8); // 1113936240.338

DO_MSLEEP(78);

// Window CWI_10 ("") created 1113936240.463

DO_MSLEEP(125);
CtxPoint(243, 205);  //1113936240.619

// Window CWI_11 ("Microsoft Word") created 1113936240.619

CtxWaitForWindowCreate(CWI_11, 156);

// Window CWI_10 ("") destroyed 1113936240.619

DO_MSLEEP(32);
CtxPoint(250, 208);  //1113936240.666

DO_MSLEEP(15);
CWI_11->setTitle("Document1 - Microsoft Word"); //1113936240.869

CtxPoint(296, 0);  //1113936245.916
```

/// This command was added to the script by clicking on the
/// ScreenUpdate message in the message panel, highlighting it,
/// then clicking on the WaitPoint button

CtxWaitForScreenUpdate(0, 0, 481, 641, 1406);

```
DO_MSLEEP(2094);
CtxPoint(56, 201);  //1113936248.260

DO_MSLEEP(250);
CtxClick(CWI_11, 47, L_BUTTON, NONE); //1113936248.307

CtxType(CWI_11, "N");  //1113936249.760

DO_MSLEEP(1453);
CtxPoint(68, 213);  //1113936249.760

CtxType(CWI_11, "ow I");  //1113936251.041
```

```
DO_MSLEEP(1281);
CtxType(CWI_11, " can type text!");  //1113936255.119

DO_MSLEEP(4078);

DO_MSLEEP(2766);

DO_MSLEEP(2094);
CtxPoint(635, 5);  //1113936260.010

DO_MSLEEP(31);
CtxClick(CWI_11, 78, L_BUTTON, NONE); //1113936260.088

CtxPoint(634, 12);  //1113936260.213

// Window CWI_12 ("Microsoft Word") created 1113936260.213

CtxWaitForWindowCreate(CWI_12, 16);

DO_MSLEEP(875);
CtxPoint(325, 275);  //1113936261.088

CtxClick(CWI_12, 63, L_BUTTON, NONE); //1113936261.151

DO_MSLEEP(62);
// Window CWI_12 ("Microsoft Word") destroyed 1113936261.213

DO_MSLEEP(16);
// Window CWI_11 ("Document1 - Microsoft Word") destroyed 1113936261.229

DO_MSLEEP(1375);
CtxPoint(385, 165);  //1113936262.963

DO_MSLEEP(359);
// Window CWI_8 ("Program Manager") destroyed 1113936262.963

// Window CWI_6 ("") destroyed 1113936262.979

// Window CWI_13 ("Please wait...") created 1113936263.244

CtxWaitForWindowCreate(CWI_13, 0);

DO_SetTransactionCleanup();

CtxDisconnect();

END_TRANSACTION();

delete CWI_1; // "Warning !!"
delete CWI_2; // "Log On to Windows"
delete CWI_3; // "Please wait..."
delete CWI_4; // "Citrix License Warning Notice"
delete CWI_5; // "UsrLogon.Cmd"
delete CWI_6; // ""
delete CWI_7; // "ICA Seamless Host Agent"
delete CWI_8; // "Program Manager"
delete CWI_9; // ""
delete CWI_10; // ""
delete CWI_11; // "Microsoft Word"
delete CWI_12; // "Microsoft Word"
delete CWI_13; // "Please wait..."

CitrixUninit();

EXIT();
```

```
   return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");

   CitrixUninit();
   EXIT();
}
```

OFS Scripts

Oracle Forms Server Script Samples

You can address specific situations or resolve certain problems by modifying converted Oracle Forms Server (OFS) scripts. The samples shown here include a description of the problem, the procedure for implementing the modification, and samples of a modified script. Script modifications are discussed for:

Parameterization of Login Credentials

Using an ICX Ticket in an OFS Script

Parameterization of Logon Credentials

Modifying the OFS Script to Parameterize Logon Credentials

If your load testing environment for Oracle Forms Server prevents a user from being logged in multiple times concurrently, you may need to parameterize the logon information. This means creating unique values for the username and password of each virtual user (VU) and transaction.

To script parameters for logon information:

1. Create a datapool that contains user names and passwords for the system you are testing.
2. Modify the script to open the datapool, read the username and password from the datapool, and use the values in the ofsSetLogonUserName() and ofsSetLogonPassWord() script commands.

Conclusion

The sample script illustrates the parameterization of logon credentials.

Sample: Parameterization of OFS Logon Credentials

The following is a sample script from an Oracle Forms Server Session that tests Oracle 10g Application Server and uses a central datapool to ensure that username and passwords are unique. The code necessary to parameterize the username and password is in bold.

Sample Script

```
/*
 *  sample-variablization.cpp
 *
 *  Script Converted on April 22, 2005 at 07:46:55 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 230
 *
 *  This script contains support for the following middlewares:
 *      - Oracle Forms Server
```

```
 */

/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Oracle Forms Server
 *  Send (Forms 6i) heartbeat every  : 4 minutes
 *  Simulate Oracle Application Login: No
 *  Stop for server error messages   : Yes
 *    Stop if server msg is matched  : No
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
extern "C" {
#include "smacro.h"
#include "do_OFS.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
    /* Declare Variables */

    //---------------------------------------------------
    // Pointers to use for username and password values
    // returned from VARDATA().
    //---------------------------------------------------
    char *username = NULL;
    char *password = NULL;

    SET_ABORT_FUNCTION(abort_function);

    DEFINE_TRANS_TYPE("sample-variablization.cpp");

    ofsSetRunOptions( "9i", OFS_HTTP, 4, OFS_CHECKMSGS );


    SYNCHRONIZE();

    BEGIN_TRANSACTION();

    //---------------------------------------------------
    // Requests that QALoad Conductor send the next
    // datapool record to the script.
    //---------------------------------------------------
    GET_DATA();

    ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.17-internal" );

    ofsHTTPSetHdrProperty("Host", "hostname.com:7778" );

    ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, "
            "*/*; q=.2" );

    ofsHTTPSetHdrProperty("Connection", "Keep-alive" );
```

```
ofsHTTPConnectToFormsServlet(
"http://hostname.com:7778/forms90/f90servlet?acceptLanguage="
        "en-us&ifcmd=startsession" );


DO_SLEEP(1);

ofsHTTPSetListenerServletParms( "?ifcmd=getinfo&ifhost=dtx76852&ifip="
        "10.15.16.128" );

ofsHTTPConnectToListenerServlet( "http://hostname.com:7778/forms90/l90servlet"
        );

 DO_SLEEP(1);

ofsHTTPSetHdrProperty("Content-type", "application/octet-stream" );

ofsHTTPInitialFormsConnect();

DO_SLEEP(1);
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "904000" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1280, 1024);
ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
 "server escapeParams=true module=test1.fmx userid=  sso_userid=%20 sso_formsid="
        "f"
 "ormsApp_ hostname.com_C49C18E49987450D8A6D907CABEFCD4C sso"
 "_subDN= sso_usrDN= debug=no host= port= buffer_records=no debug_messages=no arr"
 "ay=no obr=no query_only=no quiet=yes render=no record= tracegroup= log= term="
);
ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "0" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "255" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65535" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "4210752" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "8421504" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65280" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "12632256" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711935" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16762880" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16756655" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711680" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16777215" );
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16776960" );
ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 8, 20);
ofsSetNoRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
ofsInitSessionTimeZone( "RUNFORM", 1, OFS_ENDMSG, 530, "America/New_York" );

DO_SLEEP(3);
ofsSendRecv(1); //ClientSeqNo=1|MsgCount=1|1114170373.195

ofsSetWindowLocation( "FORMWINDOW", 6, OFS_ENDMSG, 135, 0, 0);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 750, 600);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 750, 600);

DO_SLEEP(2);
ofsSendRecv(1); //ClientSeqNo=2|MsgCount=3|1114170375.195

ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 750, 600);

DO_SLEEP(1);
```

```
    ofsSendRecv(1); //ClientSeqNo=3|MsgCount=1|1114170376.242

    //-----------------------------------------------
    // Get values from the datapool.  Column 1 contains
    // the username and column 2 the password.
    //-----------------------------------------------
    username = VARDATA(1);
    password = VARDATA(2);

    //RR__printf("V1=\"%s\"", username);
    //RR__printf("V2=\"%s\"", password);

    //-----------------------------------------------
    // Pass the username and password retrieved from
    // the datapool into the OFS script commands that
    // specify the login credentials.
    //-----------------------------------------------
    ofsSetLogonUserName( "Logon", 34, OFS_ADD, 433, username );
    ofsSetLogonPassWord( "Logon", 34, OFS_ADD, 434, password );
    //ofsSetLogonUserName( "Logon", 34, OFS_ADD, 433, "scott" );
    //ofsSetLogonPassWord( "Logon", 34, OFS_ADD, 434, "tiger" );
    ofsSetLogonDatabase( "Logon", 34, OFS_ENDMSG, 435, "dbinstance" );

    DO_SLEEP(10);
    ofsSendRecv(1); //ClientSeqNo=4|MsgCount=1|1114170385.898

    ofsActivateWindow( "WINDOW_START_APP", 11, OFS_ENDMSG, 247 );
    ofsFocus( "BUTTON", 52, OFS_ENDMSG, 174 );

    DO_SLEEP(1);
    ofsSendRecv(1); //ClientSeqNo=5|MsgCount=2|1114170387.289

    ofsShowWindow( "WINDOW_START_APP", 11, OFS_ENDMSG, 173 );

    DO_SLEEP(1);
    ofsSendRecv(1); //ClientSeqNo=6|MsgCount=1|1114170388.305

    ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 750, 600);

    DO_SLEEP(1);
    ofsSendRecv(1); //ClientSeqNo=7|MsgCount=1|1114170389.320

    ofsRemoveFocus( "BUTTON", 52, OFS_ENDMSG, 174 );
    ofsFocus( "BUTTON", 54, OFS_ENDMSG, 174 );

    DO_SLEEP(2);
    ofsSendRecv(1); //ClientSeqNo=8|MsgCount=2|1114170391.742

    ofsClickButton( "BUTTON", 54, OFS_ENDMSG, 325 );

    DO_SLEEP(1);
    ofsSendRecv(1); //ClientSeqNo=9|MsgCount=1|1114170392.773

    ofsHTTPDisconnect();

    END_TRANSACTION();
    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");
    EXIT();
}
```

Using an ICX Ticket in an OFS Script

Modifying the OFS Script to Use an ICX Ticket

Some Oracle Forms (OFS) environments use a unique cookie, called an ICX ticket, to identify sessions. When ICX tickets are present, you must make changes to the QALoad script in order to replay successfully.

To modify the script:

Modify QALoad Universal scripts to extract the ICX ticket from WWW middleware traffic and pass that value to the OFS middleware.

Conclusion

The sample script illustrates using an ICX ticket in an OFS script.

Sample: Using an ICX Ticket in an OFS Script

The following is a script sample of a Universal session WWW/OFS script. It tests an Oracle Forms 6i environment (socket mode), where an ICX Ticket cookie must be retrieved from the WWW command and used in an OFS command later in the script. In this case, the validation of the script initially fails with the following error:

Sample: Original Script

```
VU 0 : Line:494, ofsSendRecv( 2 )  //ClientSeqNo=4|MsgCount=0

VU 0 : Line: 494,  OFS-ERROR-032 -  Failed because the server sent this error message: APP-FND-01931: Your session is no longer valid or your logon information could not be reestablished from your session.

VU 0 :  Stopping HeartBeat...

VU 0 : Line 494, Exception in function ofsSendRecv:  com.compuware.qacenter.qaload.QALoadBaseScript.QALoadException: Line: 494,  OFS-ERROR-032 -  Failed because the server sent this error message: APP-FND-01931: Your session is no longer valid or your logon information could not be reestablished from your session.  at com.compuware.qacenter.qaload.Do_Webforms.RootHandler.exitOnError(Unknown Source)  at com.compuware.qacenter.qaload.Do_Webforms.SocketConnection.socketSendRecv(Unknown Source)  at com.compuware.qacenter.qaload.Do_Webforms.RunForm.sendRecv(Unknown Source)  at com.compuware.qacenter.qaload.Do_Webforms.OracleForms.ofsSendRecv(Unknown Source)

VU 0 : Line 494, EXIT
```

However, a warning is generated by QALoad during the conversion process that is put in the script:

```
/***** WARNING *****

  icx_ticket found in INITIAL_CMDLINE. Add ofsSetICXTicket() or OracleAppsLogin().

 *****/
```

There is a cookie that is needed by OFS from one of the earlier WWW requests:

```
HTTP/1.0 200 OK

Date: Fri, 22 Apr 2005 15:59:33 GMT

Server: Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_fastcgi/2.2.10 mod_oprocmgr/1.0 mod_perl/1.25

Content-Length: 4826
```

```
Set-Cookie: WF_WORKLIST_MODE=-1;path=/
```

```
Set-Cookie: bas112_vis11i=DECC7C3C58F62B42;path=/;domain=.compuware.com
```

```
Connection: close
```

```
Content-Type: text/html; charset=ISO-8859-1
```

Sample: Modified Script

Once the cookie is extracted and passed to the OFS middleware using the ofsSetICXTicket() script command, the script should replay successfully. In the example below, changes are in bold.

```
/*
 *  sample_univicx.cpp
 *
 *  Script Converted on April 22, 2005 at 10:44:58 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 230
 *
 *  This script contains support for the following middlewares:
 *      - Oracle Forms Server
 *      - WWW
 */
/* Converted using the following options:
 * General:
 *  Line Split                        : 80 characters
 *  Sleep Seconds                     : 1
 *  Auto Checkpoints                  : No
 * Oracle Forms Server
 *  Send (Forms 6i) heartbeat every   : 4 minutes
 *  Simulate Oracle Application Login: No
 *  Stop for server error messages    : Yes
 *    Stop if server msg is matched   : No
 * WWW
 *  Form Field Comments               : No
 *  Anchors as Comments               : No
 *  Client Maps as Comments           : No
 *  Debug Comments                    : No
 *  Doc Title Verification            : Yes
 *   Compare By                       : Entire Document Title
 *  Baud Rate Emulation               : No
 *  Enable Refresh                    : No
 *  Encode DBCS Characters            : No
 *  Cache                             : No
 *  Dynamic Redirect                  : Yes
 *  Dynamic Cookies                   : Yes
 *  Process Subrequests               : Yes
 *  Persistent Connections            : Yes
 *  Max Concurrent Connection         : 2
 *  Max Connection Retries            : 4
 *  Server Response Timeout           : 120
 *  HTTP Version Detection            : Auto
 *  ActiveData                        : Yes
 *  IPSpoofing                        : No
 *  Streaming Media                   : No
 *  Hostnames as IP Addresses         : No
 *  Strip All Cookies From Requests   : No
 *  Traffic Filters                   : No
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include "UniversalScript.h"

#include "do_www.h"
```

```
extern "C" {
#include "smacro.h"
#include "do_OFS.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   int i;
   char *Field[1];
   char *Anchor[1];
   char *ActionURL[1];
   char *p;
   char ICX_Ticket[100];
   char * pTicket;

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("sample_univicx.cpp");

   ofsSetRunOptions( "6i", OFS_SOCKET, 4, OFS_CHECKMSGS );

   for(i=0;i<1;i++)
      Field[i]=NULL;

   for(i=0;i<1;i++)
      Anchor[i]=NULL;

   for(i=0;i<1;i++)
      ActionURL[i]=NULL;

   DO_InitHttp(s_info);
   DO_SetTimeout(120); /* Maximum time to wait for an HTTP Reply */
   DO_SaveReplyType("text/;application/x-javascript"); /* Save replies of these types */

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();
   DO_SetMaxBrowserThreads(2);
   DO_SetMaximumRetries(4);
   DO_UsePersistentConnections(TRUE);
   DO_AutomaticSubRequests(TRUE);
   DO_DynamicCookieHandling(TRUE);
   DO_DynamicRedirectHandling(TRUE);
   DO_Cache(FALSE);     /* Disable cache */
   DO_HTTPVersion("Auto");

   /* Request: 1  */
   DO_SetCheckpointName("http://server:7779/ - chkpt:  1");
   DO_Http("GET http://server:7779/ HTTP/1.0\r\n"
          "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
          "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
          ", */*\r\n"
          "Accept-Language: en-us\r\n"
          "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
          "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
          );
```

```
    DO_VerifyDocTitle("Oracle9iAS - Welcome", TITLE);
    DO_GetAnchorHREF( "Forms 6i - Socket Mode with ICX Ticket;For Convert Option - "
            "OracleAppsLogin: URL = "
            "http://server2.com:8002/pls/vis11i/oraclemypage.home; LOGIN = "
            "mfg/welcome; PORT = 9002", &Anchor[0]);


    /* Request: 2  To: Forms 6i - Socket Mode with ICX Ticket;For Convert Option -
            OracleAppsLogin: URL =
            http://server2.com:8002/pls/vis11i/oraclemypage.home; LOGIN = mfg/welcome; PORT =
    9002 From: Oracle9iAS - Welcome */

    /*    Variable: Anchor000 links to: Forms 6i - Socket Mode with ICX Ticket;For Convert
            Option - OracleAppsLogin: URL =
            http://server2.com:8002/pls/vis11i/oraclemypage.home; LOGIN = mfg/welcome; PORT =
    9002 on page: Oracle9iAS - Welcome */
    DO_SetValue("Anchor000", Anchor[0]);
    DO_SetCheckpointName("Forms 6i - Socket Mode with ICX Ticket;For Convert Option "
            "- OracleAppsLogin: URL = "
            "http://server2.com:8002/pls/vis11i/oraclemypage.home; LOGIN = "
            "mfg/welcome; PORT = 9002 - chkpt:  2");

    DO_Http("GET {*Anchor000} HTTP/1.0\r\n"
            "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
            "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
            ", */*\r\n"
            "Referer: http://server:7779/\r\n"
            "Accept-Language: en-us\r\n"
            "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
            "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
            );
    DO_VerifyDocTitle("Oracle Applications Login", TITLE);
    DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
    DO_GetFormValueByName(FORM(1), "hidden", "rmode", 1, &Field[0]);

    /* Request: 3  From: Oracle Applications Login */
    DO_SetValue("action_statement0", ActionURL[0]);
    DO_SetValue("i_1", "mfg");
    DO_SetValue("i_2", "welcome");
    DO_SetValue("rmode", Field[0]);
    DO_SetValue("home_url", ""
            "http%3A%2F%2Fserver2.com%3A8002%2FOA_HTML%2FUS%2FICXINDEX.htm"
            );

    DO_SetCheckpointName("http://server2.com:8002/pls/vis11i/oraclemypage.home "
            "- chkpt:  3");

    DO_AdditionalSubRequest(""
"http://server2.com:8002/OA_HTML/webtools/images/toolbar_divider."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/toolbar_icon_help."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/toolbar_icon_help_active."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/toolbar_icon_help_disabled."
            "gif"
```

```
            );
    DO_AdditionalSubRequest("http://server2.com:8002/OA_MEDIA/FNDEXIT.gif"
            );
    DO_AdditionalSubRequest("http://server2.com:8002/OA_MEDIA/FNDMANOP.gif"
            );
    DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
            "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
            "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
            ", */*\r\n"
            "Referer: http://server2.com:8002/OA_HTML/US/ICXINDEX.htm\r\n"
            "Accept-Language: en-us\r\n"
            "Content-Type: application/x-www-form-urlencoded\r\n"
            "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
            "1.1.4322; .NET CLR 1.0.3705)\r\n"
            "Content-Length: {*content-length}\r\n"
            "Pragma: no-cache\r\n\r\n{i_1}&{i_2}&{rmode}&{home_url}{*crnl}"
            );

    //---------------------------------------------------------
    // Get the ICX Ticket from the HTTP headers of the response
    // to this WWW request.  Print out the value to verify the
    // value.
    //---------------------------------------------------------
    p = DO_GetUniqueString( "bas112_vis11i=", ";" );
    strcpy( ICX_Ticket, p );
    pTicket=ICX_Ticket;
    RR__printf("*****************************************************************");
    RR__printf("ICX_Ticket=\"%s\"\n", ICX_Ticket);
    RR__printf("*****************************************************************");

    //---------------------------------------------------------
    // Set the ICX ticket in the OFS middleware.  The ICX
    // ticket is used to replace the ticket contained in the
    // ofsInitSessionCmdLine() command.
    //---------------------------------------------------------
    ofsSetICXTicket(&pTicket);

    /* Request: 4  */
    DO_SetCheckpointName("http://server2.com:8002/ - chkpt:  4");
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/FNDLWAPP.gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/container_bottom_left."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/container_bottom_right."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/container_top_left_tabs."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/container_top_right_tabs."
            "gif"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/pixel_color3.gif"
```

```
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/pixel_color4.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/pixel_color6.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/pixel_gray2.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/pixel_gray5.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/tab_edit_icon.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/tab_left_non_selected."
        "gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/tab_left_selected."
        "gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/tab_right_non_selected."
        "gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/tab_right_selected."
        "gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/toolbar_left.gif"
        );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_HTML/webtools/images/toolbar_right.gif"
        );
    DO_Http("GET http://server2.com:8002/ HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: "
        "http://server2.com:8002/pls/vis11i/oraclemypage.home\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
        );
    DO_VerifyDocTitle("Oracle Applications Rapid Install", TITLE);

    /* Request: 5  */
    DO_SetValue("ICX_TICKET", "");
    DO_SetValue("RESP_APP", "INV");
    DO_SetValue("RESP_KEY", "INVENTORY");
    DO_SetValue("SECGRP_KEY", "STANDARD");
    DO_SetCheckpointName(""
        "http://server2.com:8002/pls/vis11i/fnd_icx_launch.runforms - "
```

```
            "chkpt:  5");

    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/java/awt/KeyboardFocusManager.class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/oracle/apps/fnd/jar/fndbalishare."
            "jar"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/oracle/ewt/lwAWT/BufferedAppletBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/oracle/ewt/popup/PopupAppletBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""

            "http://server2.com:8002/OA_JAVA/oracle/ewt/swing/JBufferedAppletBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""
"http://server2.com:8002/OA_JAVA/oracle/forms/engine/MainBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/sun/beans/infos/BufferedAppletBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/sun/beans/infos/JBufferedAppletBeanInfo."
            "class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/sun/beans/infos/MainBeanInfo.class"
            );
    DO_AdditionalSubRequest(""

"http://server2.com:8002/OA_JAVA/sun/beans/infos/PopupAppletBeanInfo."
            "class"
            );
    DO_Http("GET http://server2.com:8002/pls/vis11i/fnd_icx_launch.runforms?{ICX_TICKET}"
            "&{RESP_APP}&{RESP_KEY}&{SECGRP_KEY} HTTP/1.0\r\n"
            "Accept: */*\r\n"
            "Accept-Language: en-us\r\n"
            "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
            "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
            );
    DO_VerifyDocTitle("Oracle Applications 11i", TITLE);

    /* Request: 6  */
    DO_SetCheckpointName(""
            "http://server2.com:8002/OA_JAVA/oracle/forms/registry/Registry."
            "dat - chkpt:  6");

    DO_Http("GET "
            "http://server2.com:8002/OA_JAVA/oracle/forms/registry/Registry."
            "dat HTTP/1.0\r\n"
```

```
        "User-Agent: Java1.1.8.16\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );

/* Request: 7  */
DO_SetCheckpointName(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/fnd/formsClient/OracleApplications."
        "dat - chkpt:  7");

DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afapps.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afattach.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afclps.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afclpsa.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afclrall.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afclrrw.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afclsfrm.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afcopy.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afcut.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afdelrw.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afedit.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/affind.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/affldtls.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afhelp.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afinsrw.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/aflleft.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/aflright.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afnav.gif"
        );
DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afnavig.gif"
```

```
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afnxtstp.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afpaste.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afprint.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afrsp115.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afsave.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/aftrans.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afxpnd.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afxpnda.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afxpndc.gif"
        );
    DO_AdditionalSubRequest(""
        "http://server2.com:8002/OA_JAVA/oracle/apps/media/afzoom.gif"
        );
    DO_Http("GET "
        "http://server2.com:8002/OA_JAVA/oracle/apps/fnd/formsClient/OracleApplications."
        "dat HTTP/1.0\r\n"
        "User-Agent: Java1.1.8.16\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );
    ofsConnectToSocket("10.10.0.167", 9002 );
    ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "60818" );
    ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
    ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1280, 1024);

    /***** WARNING *****
      icx_ticket found in INITIAL_CMDLINE. Add ofsSetICXTicket() or OracleAppsLogin().
     *****/
    ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
     "server module=/oracle/appl/vis11iappl/fnd/11.5.0/forms/US/FNDSCSGN userid=APPLS"
     "YSPUB/PUB@vis11i fndnam=APPS  config='bas112_vis11i' icx_ticket='.903869489' re"
     "sp='INV/INVENTORY' secgrp='STANDARD' "
    );
    ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "0" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "255" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65535" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "4210752" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "8421504" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65280" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "12632256" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711935" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16762880" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16756655" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711680" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16777215" );
    ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16776960" );
    ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
```

```
ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 9, 18);
ofsSetRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
ofsSetPropertyString("RUNFORM", 1, OFS_ADD, 510,
 "NLS_LANG='AMERICAN_AMERICA.WE8ISO8859P1' FORMS60_USER_DATE_FORMAT="
        "'DD-MON-RRRR'"
 " FORMS60_USER_DATETIME_FORMAT='DD-MON-RRRR HH24:MI:SS' NLS_DATE_LANGUAGE='AMERI"
 "CAN' NLS_SORT='BINARY' NLS_NUMERIC_CHARACTERS='.,'"
);
ofsInitSessionTimeZone( "RUNFORM", 1, OFS_ENDMSG, 527, "EST" );
ofsSendRecv(1); //ClientSeqNo=1|MsgCount=1|1114181041.445

ofsSetWindowLocation( "FORMWINDOW", 6, OFS_ENDMSG, 135, 0, 0);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 250, 120);
ofsSetWindowLocation( "Object Properties", 11, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Progress", 15, OFS_ENDMSG, 135, 288, 336);
ofsSetWindowLocation( "Rename Label", 20, OFS_ENDMSG, 135, 192, 192);
ofsSetWindowLocation( "Launch", 24, OFS_ENDMSG, 135, 192, 192);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 250, 120);
ofsSetPropertyInteger( "CUSTOMCONTROL", 54, OFS_ADD, 1, "21002" );
ofsSetPropertyInteger( "CUSTOMCONTROL", 54, OFS_ADD, 1, "0" );
ofsSetPropertyStringArray( "CUSTOMCONTROL", 54, OFS_ENDMSG, 1, "0" );
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1280, 1024);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1280, 1024);
ofsSendRecv(1); //ClientSeqNo=2|MsgCount=10|1114181041.445

ofsSendRecv(2); //ClientSeqNo=3|MsgCount=0|1114181041.633

ofsSendRecv(2); //ClientSeqNo=4|MsgCount=0|1114181043.961

ofsSetSelection( "TEXTFIELD", 89, OFS_ADD, 195, 0, 0);
ofsSetCursorPosition( "TEXTFIELD", 89, OFS_ENDMSG, 193, "0" );
ofsSetWindowSize( "NAVIGATOR", 28, OFS_ENDMSG, 137, 670, 432);
ofsSetWindowSize( "NAVIGATOR", 28, OFS_ENDMSG, 137, 670, 600);
ofsSetWindowLocation( "NAVIGATOR", 28, OFS_ENDMSG, 135, 55, 0);
ofsSetWindowLocation( "NAVIGATOR", 28, OFS_ENDMSG, 135, 55, 46);
ofsShowWindow( "NAVIGATOR", 28, OFS_ENDMSG, 173 );
ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsSendRecv(2); //ClientSeqNo=5|MsgCount=8|1114181044.258

ofsSendRecv(2); //ClientSeqNo=6|MsgCount=0|1114181045.305

ofsSetSelection( "TEXTFIELD", 89, OFS_ENDMSG, 195, 0, 0);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1280, 1024);
ofsSendRecv(1); //ClientSeqNo=7|MsgCount=2|1114181045.320

ofsRemoveFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsFocus( "TLIST", 95, OFS_ENDMSG, 174 );
ofsSendRecv(1); //ClientSeqNo=8|MsgCount=2|1114181058.445


DO_SLEEP(13);
ofsListItemValue( "TLIST", 95, OFS_ENDMSG, 131, "0" );  /*Item value = 1. View Quantities
        On-hand*/

ofsSendRecv(1); //ClientSeqNo=9|MsgCount=1|1114181058.555

ofsRemoveFocus( "TLIST", 95, OFS_ENDMSG, 174 );
ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsSendRecv(1); //ClientSeqNo=10|MsgCount=2|1114181058.664

ofsRemoveFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
```

```
ofsFocus( "TLIST", 95, OFS_ENDMSG, 174 );
ofsActivateListItem( "TLIST", 95, OFS_ENDMSG, 341, "0" );
ofsSendRecv(1); //ClientSeqNo=11|MsgCount=3|1114181060.086

ofsRemoveFocus( "TLIST", 95, OFS_ENDMSG, 174 );
ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsSendRecv(2); //ClientSeqNo=12|MsgCount=2|1114181060.492

ofsSetSelection( "TEXTFIELD", 269, OFS_ADD, 195, 0, 0);
ofsSetCursorPosition( "TEXTFIELD", 269, OFS_ENDMSG, 193, "0" );
ofsRemoveFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsFocus( "TEXTFIELD", 269, OFS_ENDMSG, 174 );
ofsDeActivateWindow( "Find On_hand Quantities", 161, OFS_ENDMSG, 247 );
ofsSendRecv(2); //ClientSeqNo=13|MsgCount=4|1114181060.695

ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 1, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 2, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 3, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 4, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 5, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 6, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 7, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 8, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 9, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 10, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 11, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 12, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 13, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 14, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 15, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 16, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 17, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 451, 18, 1);
ofsSendRecv(1); //ClientSeqNo=14|MsgCount=18|1114181061.289

ofsSetWindowLocation( "Object Properties", 124, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Sort Data", 128, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Calendar", 132, OFS_ENDMSG, 135, 96, 96);
ofsSetWindowLocation( "Progress", 136, OFS_ENDMSG, 135, 288, 336);
ofsSetWindowLocation( "Autosize All", 141, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Folder Query", 145, OFS_ENDMSG, 135, 240, 216);
ofsSetWindowLocation( "Save Folder", 149, OFS_ENDMSG, 135, 269, 307);
ofsSetWindowLocation( "Change Prompt", 153, OFS_ENDMSG, 135, 269, 326);
ofsSetWindowLocation( "Folder Tools", 157, OFS_ENDMSG, 135, 240, 144);
ofsSendRecv(1); //ClientSeqNo=15|MsgCount=9|1114181061.289

ofsActivateWindow( "Find On_hand Quantities", 161, OFS_ENDMSG, 247 );
ofsLOVSelection( "LISTVALUESDIALOG", 274, OFS_ENDMSG, 450, "3" );
ofsSendRecv(1); //ClientSeqNo=16|MsgCount=2|1114181065.758


DO_SLEEP(4);

DO_SLEEP(7);
ofsSendRecv(2); //ClientSeqNo=17|MsgCount=0|1114181075.195

ofsSetSelection( "TEXTFIELD", 269, OFS_ADD, 195, 0, 3);
ofsSetCursorPosition( "TEXTFIELD", 269, OFS_ENDMSG, 193, "3" );
ofsSetWindowLocation( "Item On_hand Quantities", 173, OFS_ENDMSG, 135, 29, 29);

ofsSetWindowLocation( "Revision On_hand Quantities", 177, OFS_ENDMSG, 135, 58,
        58);

ofsSetWindowLocation( "Subinventory On_hand Quantities", 181, OFS_ENDMSG, 135, 87
        , 87);
```

```
ofsSetWindowLocation( "Detailed On_hand Quantities", 185, OFS_ENDMSG, 135, 116,
        116);

ofsSetWindowLocation( "Lot On_hand Quantities", 169, OFS_ENDMSG, 135, 145, 145);

ofsSetWindowLocation( "Serial On_hand Quantities", 165, OFS_ENDMSG, 135, 174,
        174);

ofsCloseWindow( "Find On_hand Quantities", 161, OFS_ENDMSG, 216 );

DO_SLEEP(2);
ofsSendRecv(1); //ClientSeqNo=18|MsgCount=8|1114181077.570

ofsFocus( "BUTTON", 73, OFS_ENDMSG, 174 );
ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsRemoveFocus( "BUTTON", 73, OFS_ENDMSG, 174 );
ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );

DO_SLEEP(1);
ofsSendRecv(1); //ClientSeqNo=19|MsgCount=4|1114181078.930

ofsCloseWindow( "NAVIGATOR", 28, OFS_ENDMSG, 216 );
ofsSendRecv(1); //ClientSeqNo=20|MsgCount=1|1114181080.102

ofsDeActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsSendRecv(1); //ClientSeqNo=21|MsgCount=1|1114181080.102


DO_SLEEP(1);
ofsSetPropertyInteger( "CUSTOMCONTROL", 293, OFS_ADD, 1, "20002" );
ofsSetPropertyInteger( "CUSTOMCONTROL", 293, OFS_ENDMSG, 1, "0" );
ofsSendRecv(1); //ClientSeqNo=22|MsgCount=1|1114181080.680

ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsSendRecv(2); //ClientSeqNo=23|MsgCount=1|1114181080.867

ofsSetPropertyInteger( "FORMWINDOW", 6, OFS_ENDMSG, 130, "1" );
ofsSendRecv(3); //ClientSeqNo=24|MsgCount=1|1114181080.867

ofsSocketDisconnect();

DO_SetTransactionCleanup();
/* Clear up some internal storage used for DO_SetValue() */
DO_HttpCleanup();
for(i=0; i<1; i++)
{
free(Field[i]);
Field[i]=NULL;
}
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
for(i=0; i<1; i++)
{
free(ActionURL[i]);
ActionURL[i]=NULL;
}

END_TRANSACTION();
DO_FreeHttp();

EXIT();
```

```
   return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");
   DO_FreeHttp();

   EXIT();
}
```

Using HTTP Cookies and ICX Tickets in an OFS Script

Using HTTP Cookies and ICX Tickets in an OFS Script

Some Oracle Forms environments use a unique cookies to identify sessions requiring changes to be made to the QALoad script in order to replay successfully.

Scripting

In cases like these, QALoad Universal scripts need to be modified to extract the ICX ticket and any other required cookies from WWW middleware traffic and to pass them to the OFS middleware.

Example

The following is a script sample of a Universal session WWW/OFS script testing an Oracle Forms 6i environment (servlet mode) where an ICX Ticket cookie needs to be retrieved from the WWW command and used in an OFS command later in the script.

This server environment has two cookies that must be retrieved from the WWW traffic in order to pass it into the OFS middleware: JServSessionIdrootbas114 and nvis510. Once these cookies are extracted and passed to the OFS middleware, the script should replay successfully.

In the example below, changes are shown in bold.

```
/*
 *  This script is a sample Universal OFS (Servlet mode) / WWW script
 *  that shows examples of setting cookies and setting the ICX ticket.
 *
 *  This script contains support for the following middlewares:
 *      - Oracle Forms Server
 *      - WWW
 */
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : No
 * Oracle Forms Server
 *  Simulate Oracle Application Login: No
 *  Stop for server error messages   : Yes
 *    Stop if server msg is matched  : No
 * WWW
```

```
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Enable Refresh                   : No
 *  Represent CJK as Octal Characters: No
 *  Cache                            : No
 *  Dynamic Redirect                 : No
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 2
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version                     : 1.1
 *  Proxy HTTP Version               : 1.0
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 *  Traffic Filters                  : No
 */
#define SCRIPT_VER 0x00000505UL
#include <stdio.h>
#include "UniversalScript.h"
#include "do_www.h"
extern "C" {
#include "smacro.h"
#include "do_OFS.h"
}
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
```

```
extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */
   int i;
   char *Field[3];
   char *Anchor[2];
   char *ActionURL[1];
//-------------------------------------------------------------------------------
// Variable declarations
//-------------------------------------------------------------------------------
   char *sessionid = NULL; // JServSessionIdbas114 cookie
   char *nvis510 = NULL; // nvis510 cookie
   char *cookieHeaderValue = NULL; // combined cookie
   char *icx_ticket = NULL;
   char buf[512]; // used for formatting of cmdline,
   //  URL parameters

   SET_ABORT_FUNCTION(abort_function);
   DEFINE_TRANS_TYPE("sample_cookiesandicxticket.cpp");
   ofsSetRunOptions(FORMS_6i_11i_SERVLET, OFS_HTTP, 4, OFS_CHECKMSGS);
   for(i=0;i<3;i++)
   Field[i]=NULL;
   for(i=0;i<2;i++)
   Anchor[i]=NULL;
   for(i=0;i<1;i++)
   ActionURL[i]=NULL;
   DO_InitHttp(s_info);
   DO_SetTimeout(120); /* Maximum time to wait for an HTTP Reply */
   DO_SaveReplyType("text/;application/x-javascript"); /* Save replies of these types */
   SET_SCRIPT_LANGUAGE(SLID_English);

   SYNCHRONIZE();
   /* Select following statement for reuse of Session ID with SSL */
   /* If session ID needs only to be reused within a transaction, insert */
   /* after the BEGIN_TRANSACTION statement */
   /* DO_SSLReuseSession(TRUE); */

   BEGIN_TRANSACTION();
   DO_SetTransactionStart();
```

```
DO_SetMaxBrowserThreads(2);
DO_SetMaximumRetries(4);
DO_UsePersistentConnections(TRUE);
DO_AutomaticSubRequests(TRUE);
DO_DynamicCookieHandling(TRUE);
DO_DynamicRedirectHandling(FALSE);
DO_Cache(FALSE);     /* Disable cache */
DO_HTTPVersion("1.1");
DO_ProxyHTTPVersion("1.0");


/* Request: 1  */
DO_Http("GET http://server.company.com:8060/oa_servlets/AppsLogin HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
        );

/* Request: 2  */
DO_SetValue("requestUrl", "APPSHOMEPAGE");
DO_SetValue("cancelUrl", ""
        "http%3A%2F%2Fserver.company.com%3A8060%2Foa_servlets%2Foracle.apps.fnd."
        "sso.AppsLogin");

DO_Http("GET http://server.company.com:8060/OA_HTML/AppsLocalLogin.jsp?{requestUrl}"
        "&{cancelUrl} HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
        );
DO_VerifyDocTitle("Login", TITLE);
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
DO_GetFormValueByName(FORM(1), "hidden", "langCode", 1, &Field[0]);
DO_GetFormValueByName(FORM(1), "hidden", "cancelUrl", 1, &Field[1]);
DO_GetFormValueByName(FORM(1), "hidden", "requestUrl", 1, &Field[2]);
```

```
/* Request: 3  */
DO_Http("GET http://server.company.com:8060/OA_HTML/cabo/images/cache/cghec.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/AppsLocalLogin.jsp?requestUrl="
        "APPSHOMEPAGE&cancelUrl="
        "http%3A%2F%2Fserver.company.com%3A8060%2Foa_servlets%2Foracle.apps.fnd."
        "sso.AppsLogin\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
        );

DO_SLEEP(8);

/* Request: 4  */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("username", "operations");
DO_SetValue("password", "welcome");
DO_SetValue("langCode", Field[0]);
DO_SetValue("cancelUrl", Field[1]);
DO_SetValue("requestUrl", Field[2]);
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/AppsLocalLogin.jsp?requestUrl="
        "APPSHOMEPAGE&cancelUrl="
        "http%3A%2F%2Fserver.company.com%3A8060%2Foa_servlets%2Foracle.apps.fnd."
        "sso.AppsLogin\r\n"
        "Accept-Language: en-us\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Content-Length: {*content-length}\r\n"
        "Pragma: no-
cache\r\n\r\n{username}&{password}&{langCode}&{cancelUrl}&{requestUrl}"
        );

//----------------------------------------------------------------------------
```

```
// There are a couple of cookies that need to be extracted from the response
// to the POST request done for the login.  In this case there are two cookies
// (JServSessionIdrootbas114 and nvis510) that need to be combined into
// a single cookie header in the format "Set-Cookie: name=value; name=value".
// The resulting string is stored in cookieHeaderValue which is used later.
//-----------------------------------------------------------------------
//RR__printf("REPLY: %s\n", DO_GetReplyBuffer());
    sessionid = (char *)malloc(128);
    memset(sessionid, 0, 128);
    strcpy(sessionid, "JServSessionIdrootbas114=");
    strcat(sessionid, DO_GetUniqueString("Set-Cookie: JServSessionIdrootbas114=", ";"));
// RR__printf("COOKIE: %s", sessionid);
    nvis510 = (char *)malloc(128);
    memset(nvis510, 0, 128);
    strcpy(nvis510, "nvis510=");
    strcat(nvis510, DO_GetUniqueString("Set-Cookie: nvis510=", ";"));
// RR__printf("COOKIE: %s", nvis510);
    cookieHeaderValue = (char *)malloc(512);
    memset(cookieHeaderValue, 0, 512);
    strcpy(cookieHeaderValue, sessionid);
    strcat(cookieHeaderValue, "; ");
    strcat(cookieHeaderValue, nvis510);
//RR__printf("COOKIE: %s", cookieHeaderValue);
    DO_SLEEP(9);

    /* Request: 5  */
    DO_SetValue("OAFunc", "OAHOMEPAGE");
    DO_Http("GET http://server.company.com:8060/OA_HTML/OA.jsp?{OAFunc} "
            "HTTP/1.0\r\n"
            "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
            "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
            ", */*\r\n"
             "Referer: http://server.company.com:8060/OA_HTML/AppsLocalLogin.jsp?requestUrl="
        "APPSHOMEPAGE&cancelUrl="
"http%3A%2F%2Fserver.company.com%3A8060%2Foa_servlets%2Foracle.apps.fnd."
        "sso.AppsLogin\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Pragma: no-cache\r\n\r\n"
```

```
        );
    DO_VerifyDocTitle("Oracle Applications Home Page", TITLE);
    DO_GetAnchorHREF( "Interaction History JSP Admin", &Anchor[0]);

    /* Request: 6  From: Oracle Applications Home Page */
    DO_Http("GET http://server.company.com:8060/OA_HTML/cabo/images/cache/cghec.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
        "OAHOMEPAGE\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

DO_SLEEP(18);

/* Request: 7  To: Interaction History JSP Admin From: Oracle Applications Home
        Page */


/*    Variable: Anchor000 links to: Interaction History JSP Admin on page: Oracle
        Applications Home Page */

DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
        "OAHOMEPAGE\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );
DO_VerifyDocTitle("Oracle Applications Home Page", TITLE);
DO_GetAnchorHREF( "Inventory, Vision Operations (USA)", &Anchor[1]);

/* Request: 8  From: Oracle Applications Home Page */
```

```
DO_Http("GET http://server.company.com:8060/OA_HTML/cabo/images/cache/cghec.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
        "OAHOMEPAGE&akRegionApplicationId=0&navRespId=23216&navRespAppId=690&navSecGrpId="
        "0&transactionid=1703444656&oapc=2&oas=EltFim-hJGa3FZYVV1DbQg..\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

DO_SLEEP(9);

/* Request: 9  To: Inventory, Vision Operations (USA) From: Oracle Applications
        Home Page */



/*    Variable: Anchor001 links to: Inventory, Vision Operations (USA) on page:
        Oracle Applications Home Page */

DO_SetValue("Anchor001", Anchor[1]);
DO_Http("GET {*Anchor001} HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
        "OAHOMEPAGE&akRegionApplicationId=0&navRespId=23216&navRespAppId=690&navSecGrpId="
        "0&transactionid=1703444656&oapc=2&oas=EltFim-hJGa3FZYVV1DbQg..\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );
DO_VerifyDocTitle("Oracle Applications Home Page", TITLE);

  /* Request: 10  From: Oracle Applications Home Page */
  DO_Http("GET http://server.company.com:8060/OA_HTML/cabo/images/cache/cghec.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
```

```
        "OAHOMEPAGE&akRegionApplicationId=0&navRespId=50583&navRespAppId=401&navSecGrpId="
        "0&transactionid=1703444656&oapc=3&oas=uO79v_OeLsXVyAHNb2CnuQ..\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

  DO_SLEEP(10);

  /* Request: 11  From: Oracle Applications Home Page */
  DO_SetValue("function_id", "1006901");
  DO_SetValue("resp_id", "50583");
  DO_SetValue("resp_appl_id", "401");
  DO_SetValue("security_group_id", "0");
  DO_SetValue("lang_code", "US");
  DO_SetValue("oas", "cmiXz7a5U71yEfvDNQpeyQ..");
  DO_SetValue("formsLink", "yes");
  DO_Http("GET http://server.company.com:8060/OA_HTML/RF.jsp?{function_id}&{resp_id}"
        "&{resp_appl_id}&{security_group_id}&{lang_code}&{oas}&{formsLink} "
        "HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
        "OAHOMEPAGE&akRegionApplicationId=0&navRespId=50583&navRespAppId=401&navSecGrpId="
        "0&transactionid=1703444656&oapc=3&oas=uO79v_OeLsXVyAHNb2CnuQ..\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

  DO_SLEEP(1);

  /* Request: 12  From: Oracle Applications Home Page */
  DO_SetValue("resp_app", "INV");
  DO_SetValue("resp_key", "INVENTORY_VISION_OPERATIONS");
  DO_SetValue("secgrp_key", "STANDARD");
  DO_SetValue("start_func", "INV_INVMATWB");
  DO_SetValue("other_params", "");
```

```
DO_Http("GET http://server.company.com:8060/pls/nvis510/fnd_icx_launch.launch?{resp_app}"
      "&{resp_key}&{secgrp_key}&{start_func}&{other_params} HTTP/1.0\r\n"
      "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
      "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
      ", */*\r\n"
      "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
      "OAHOMEPAGE&akRegionApplicationId=0&navRespId=50583&navRespAppId=401&navSecGrpId="
      "0&transactionid=1703444656&oapc=3&oas=uO79v_OeLsXVyAHNb2CnuQ..\r\n"
      "Accept-Language: en-us\r\n"
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
      "1.1.4322; .NET CLR 1.0.3705)\r\n"
      "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
      );

/* Request: 13  From: Oracle Applications Home Page */
DO_SetValue("appletmode", "nonforms");
DO_SetValue("HTMLpageTitle", "");
DO_SetValue("HTMLpreApplet", "");
DO_SetValue("code", "oracle/apps/fnd/formsClient/FormsLauncher.class");
DO_SetValue("width", "400");
DO_SetValue("height", "100");
DO_SetValue("archive", "/OA_JAVA/oracle/apps/fnd/jar/fndforms.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndformsi18n.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndewt.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndswing.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndbalishare.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndaol.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndctx.jar"
      ",/OA_JAVA/oracle/apps/fnd/jar/fndlist.jar");

DO_SetValue("gp14", "jinit_appletcache");
DO_SetValue("gv14", "offjinit_appletcache=off");
DO_SetValue("gp2", "resp_app");
DO_SetValue("gv2", "INV");
DO_SetValue("gp3", "resp");
DO_SetValue("gv3", "INVENTORY_VISION_OPERATIONS");
DO_SetValue("gp4", "sec_group");
DO_SetValue("gv4", "STANDARD");
DO_SetValue("gp5", "function");
DO_SetValue("gv5", "INV_INVMATWB");
DO_SetValue("gp6", "other_params");
```

```
  DO_SetValue("gv6", "");
  DO_SetValue("gp7", "forms_url");
  DO_SetValue("gv7", ""
        "http%3A%2F%2Fserver.company.com%3A8060%2Fpls%2Fnvis510%2Ffnd_icx_launch."
        "runforms%3FICX_TICKET%3D%26resp_app%3DINV%26resp_key%3DINVENTORY_VISION_OPERATIONS%
"
        "26secgrp_key%3DSTANDARD%26start_func%3DINV_INVMATWB%26other_params%3D"
        );

  DO_SetValue("encoding", "UTF-8");
  DO_SetValue("gp8", "error_url");
  DO_SetValue("gv8", ""
        "http%3A%2F%2Fserver.company.com%3A8060%2FOA_HTML%2Fjsp%2Ffnd%2Ffnderror."
        "jsp%3Fdbc%3Dnvis510");

  DO_SetValue("gp12", "port");
  DO_SetValue("gv12", "6945");
  DO_SetValue("gp13", "dbc");
  DO_SetValue("gv13", "nvis510");
  DO_SetValue("gp15", "icx_ticket");
  DO_SetValue("gv15", "614175610");
  DO_Http("GET http://server.company.com:8060/dev60cgi/f60cgi?&{appletmode}&{HTMLpageTitle}"
          "&{HTMLpreApplet}&{code}&{width}&{height}&{archive}&{gp14}&{gv14}&{gp2}&{gv2}"
        "&{gp3}&{gv3}&{gp4}&{gv4}&{gp5}&{gv5}&{gp6}&{gv6}&{gp7}&{gv7}&{encoding}"
        "&{gp8}&{gv8}&{gp12}&{gv12}&{gp13}&{gv13}&{gp15}&{gv15} HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
        "application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword"
        ", */*\r\n"
        "Referer: http://server.company.com:8060/OA_HTML/OA.jsp?OAFunc="
          "OAHOMEPAGE&akRegionApplicationId=0&navRespId=50583&navRespAppId=401&navSecGrpId="
        "0&transactionid=1703444656&oapc=3&oas=uO79v_OeLsXVyAHNb2CnuQ..\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

  /* Request: 14  */
  DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/java/awt/KeyboardFocusManager."
        "class HTTP/1.0\r\n"
```

```
        "Accept: */*\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );
DO_VerifyDocTitle("403 Forbidden", TITLE);

/* Request: 15  */
DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/java/awt/KeyboardFocusManager."
        "class HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );
DO_VerifyDocTitle("403 Forbidden", TITLE);

/* Request: 16  */
DO_Http("GET http://server.company.com:8060/OA_MEDIA/appslogo_new.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );

/* Request: 17  */
DO_Http("GET http://server.company.com:8060/OA_MEDIA/appslogo_new.gif "
        "HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );

DO_SLEEP(2);

/* Request: 18  */
DO_SetValue("ICX_TICKET", "");
DO_SetValue("resp_app", "INV");
DO_SetValue("resp_key", "INVENTORY_VISION_OPERATIONS");
DO_SetValue("secgrp_key", "STANDARD");
```

```
  DO_SetValue("start_func", "INV_INVMATWB");
  DO_SetValue("other_params", "");
  DO_Http("GET 
  http://server.company.com:8060/pls/nvis510/fnd_icx_launch.runforms?{ICX_TICKET}"
        "&{resp_app}&{resp_key}&{secgrp_key}&{start_func}&{other_params} "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Accept-Language: en-us\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
        );

//--------------------------------------------------------------------------------
// The ICX ticket needs to be extracted from the response from the
// fnd_icx_launch.runforms GET request
//--------------------------------------------------------------------------------
//RR__printf("REPLY: %s\n", DO_GetReplyBuffer());
  icx_ticket = (char*) malloc(128);
  strcpy(icx_ticket, DO_GetUniqueString("icx_ticket='", "'"));
// RR__printf("ICX TICKET: \"%s\"\n", icx_ticket);
  sprintf(buf, "+config='nvis510'+icx_ticket='%s'+resp="
        "'INV%2FINVENTORY_VISION_OPERATIONS'+secgrp='STANDARD'+start_func="
        "'INV_INVMATWB'", icx_ticket);

  DO_SetValue("lookAndFeel", "ORACLE");
  DO_SetValue("colorScheme", "BLAF");
  DO_SetValue("lang", "US");
  DO_SetValue("env", "NLS_LANG='AMERICAN_AMERICA.UTF8'+FORMS60_USER_DATE_FORMAT="
        "'DD-MON-RRRR'+FORMS60_USER_DATETIME_FORMAT="
        "'DD-MON-RRRR%20HH24%3AMI%3ASS'+NLS_DATE_LANGUAGE='AMERICAN'+NLS_SORT="
        "'BINARY'+NLS_NUMERIC_CHARACTERS='.,'");

//--------------------------------------------------------------------------------
// The ICX ticket value extracted needs to go into the URL parameters
//--------------------------------------------------------------------------------
// DO_SetValue("form_params",
//                  "+config='nvis510'+icx_ticket='.1315587967'+resp="
//         "'INV%2FINVENTORY_VISION_OPERATIONS'+secgrp='STANDARD'+start_func="
//           "'INV_INVMATWB'");
```

```
DO_SetValue("form_params", buf);

DO_SetValue("encoding", "UTF-8");
DO_Http("GET "http://server.company.com:8060/dev60cgi/f60cgi?{lookAndFeel}&{colorScheme}"
      "&{lang}&{env}&{form_params}&{encoding} HTTP/1.0\r\n"
      "Accept: */*\r\n"
      "Accept-Language: en-us\r\n"
      "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
      "1.1.4322; .NET CLR 1.0.3705)\r\n\r\n"
      );
DO_VerifyDocTitle("Oracle Applications 11i", TITLE);

/* Request: 20  */
DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/oracle/forms/icons/oracle_logo_light."
      "gif HTTP/1.0\r\n"
      "Accept: */*\r\n"
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
      "1.1.4322; .NET CLR 1.0.3705)\r\n"
      "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
      );
DO_VerifyDocTitle("404 Not Found", TITLE);

/* Request: 21  */
DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/oracle/forms/icons/oracle_logo_light."
      "gif HTTP/1.1\r\n"
      "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
      "User-Agent: Java1.3.1.18-internal\r\n"
      "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
      );
DO_VerifyDocTitle("404 Not Found", TITLE);

/* Request: 22  */
DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/oracle/forms/icons/oracle_logo_light."
      "gif HTTP/1.0\r\n"
      "Accept: */*\r\n"
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
      "1.1.4322; .NET CLR 1.0.3705)\r\n"
```

```
        "Cookie:  oracle.uix=0^^^^GMT-4:00; \r\n\r\n"
        );
  DO_VerifyDocTitle("404 Not Found", TITLE);

  /* Request: 23  */
  DO_Http("GET "
          "http://server.company.com:8060/OA_JAVA/oracle/forms/icons/oracle_logo_light."
        "gif HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );
  DO_VerifyDocTitle("404 Not Found", TITLE);

  DO_SLEEP(2);

  /* Request: 24  */
  DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/oracle/forms/registry/Registry."
        "dat HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );

  DO_SLEEP(1);

  /* Request: 25  */
  DO_Http("GET "
          "http://server.company.com:8060/OA_JAVA/oracle/apps/fnd/formsClient/OracleApplicati
  ons."
        "dat HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );

  ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.18-internal" );

  ofsHTTPSetHdrProperty("Host", "server.company.com:8060" );
```

```
  ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, "
          "*/*; q=.2" );

  ofsHTTPSetHdrProperty("Connection", "Keep-alive" );

//----------------------------------------------------------------------------
// Tell the OFS middleware to set the Cookie header for the requests that
// follow...
//----------------------------------------------------------------------------
  ofsHTTPSetHdrProperty("cookie:", cookieHeaderValue);

  ofsHTTPConnectToFormsServlet( "http://server.company.com:8060/forms/formservlet?ifcmd="
          "getinfo&ifhost=dtx76852&ifip=10.15.16.128" );


  DO_SLEEP(2);

  ofsHTTPSetHdrProperty("Content-type", "application/octet-stream" );

  ofsHTTPInitialFormsConnect();


  DO_SLEEP(2);
  ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "60824" );
  ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
  ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1600, 1200);

//----------------------------------------------------------------------------
// The ICX ticket value extracted from the web traffic needs to be passed to
// OFS replay as part of the cmdline
//----------------------------------------------------------------------------
sprintf(buf,
 "server module=/oracle/appl/nvis510appl/fnd/11.5.0/forms/US/FNDSCSGN fndnam=APPS"
  "  config='nvis510' icx_ticket='%s' resp='INV/INVENTORY_VISION_OPERATIO"
  "NS' secgrp='STANDARD' start_func='INV_INVMATWB'",
icx_ticket);
// ORIGINAL CODE:
// ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
//  "server module=/oracle/appl/nvis510appl/fnd/11.5.0/forms/US/FNDSCSGN fndnam=APPS"
//  "  config='nvis510' icx_ticket='.1315587967' resp='INV/INVENTORY_VISION_OPERATIO"
```

```
//  "NS' secgrp='STANDARD' start_func='INV_INVMATWB'"
// );
  ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265, buf);
  ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "0" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "255" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65535" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "4210752" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "8421504" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "65280" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "12632256" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711935" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16762880" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16756655" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16711680" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16777215" );
  ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16776960" );
  ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
  ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
  ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
  ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
  ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 8, 22);
  ofsSetRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
  ofsSetPropertyString("RUNFORM", 1, OFS_ADD, 510,
   "NLS_LANG='AMERICAN_AMERICA.UTF8' FORMS60_USER_DATE_FORMAT='DD-MON-RRRR' FORMS60"
   "_USER_DATETIME_FORMAT='DD-MON-RRRR HH24:MI:SS' NLS_DATE_LANGUAGE='AMERICAN' NLS"
   "_SORT='BINARY' NLS_NUMERIC_CHARACTERS='.,'"
  );

  ofsSetPropertyString( "RUNFORM", 1, OFS_ENDMSG, 527, "America/New_York" );
  DO_SLEEP(2);

  DO_SLEEP(2);
  ofsSendRecv(1); //ClientSeqNo=1|MsgCount=1|1128450767.703



  DO_SLEEP(1);

  /* Request: 26  */
  DO_Http("GET "
```

```
        "http://server.company.com:8060/OA_JAVA/oracle/apps/media/afclrall.gif "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR "
        "1.1.4322; .NET CLR 1.0.3705)\r\n"
        "Cookie:  oracle.uix=0^^^^GMT-4:00; JServSessionIdforms="
        "z6m86mb2u1.mA5Nah4OcA9vqN4Lb6yImQXHq7jRml9z/AbJphCLc3mKa0--; \r\n\r\n"
        );

/* Request: 27  */
DO_Http("GET "
        "http://server.company.com:8060/OA_JAVA/oracle/apps/media/afclrall.gif "
        "HTTP/1.1\r\n"
        "cookie:  oracle.uix=0^^^^GMT-4:00; JServSessionIdforms="
        "z6m86mb2u1.mA5Nah4OcA9vqN4Lb6yImQXHq7jRml9z/AbJphCLc3mKa0--; \r\n"
        "User-Agent: Java1.3.1.18-internal\r\n"
        "Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2\r\n\r\n"
        );
ofsSetWindowLocation( "FORMWINDOW", 6, OFS_ENDMSG, 135, 0, 0);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 558, 120);
ofsSetWindowLocation( "Object Properties", 11, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Progress", 15, OFS_ENDMSG, 135, 288, 336);
ofsSetWindowLocation( "Rename Label", 20, OFS_ENDMSG, 135, 192, 192);
ofsSetWindowLocation( "Launch", 24, OFS_ENDMSG, 135, 192, 192);
ofsSetPropertyInteger( "CUSTOMCONTROL", 54, OFS_ADD, 1, "21002" );
ofsSetPropertyInteger( "CUSTOMCONTROL", 54, OFS_ADD, 1, "0" );
ofsSetPropertyStringArray( "CUSTOMCONTROL", 54, OFS_ENDMSG, 1, "0" );
ofsSendRecv(1); //ClientSeqNo=2|MsgCount=7|1128450772.156


DO_SLEEP(2);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 558, 120);

ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1600, 1200);
DO_SLEEP(2);
ofsSendRecv(2); //ClientSeqNo=3|MsgCount=2|1128450775.531


DO_SLEEP(1);
ofsSetCursorPosition( "TEXTFIELD", 89, OFS_ENDMSG, 193, "0" );
ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
```

```
ofsShowWindow( "NAVIGATOR", 28, OFS_ENDMSG, 173 );

ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
DO_SLEEP(3);
ofsSendRecv(2); //ClientSeqNo=4|MsgCount=4|1128450779.859


DO_SLEEP(1);
ofsSetWindowSize( "NAVIGATOR", 28, OFS_ENDMSG, 137, 819, 432);
ofsSetWindowSize( "NAVIGATOR", 28, OFS_ENDMSG, 137, 819, 570);

ofsSetWindowLocation( "NAVIGATOR", 28, OFS_ENDMSG, 135, 0, 13);
DO_SLEEP(2);
ofsSendRecv(2); //ClientSeqNo=5|MsgCount=3|1128450781.562

ofsSendRecv(2); //ClientSeqNo=6|MsgCount=0|1128450783.375


DO_SLEEP(2);

DO_SLEEP(2);
ofsSendRecv(2); //ClientSeqNo=7|MsgCount=0|1128450786.875


DO_SLEEP(1);
ofsSetWindowLocation( "Object Properties", 125, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Sort Data", 133, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Folder Tools", 137, OFS_ENDMSG, 135, 240, 144);
ofsSetWindowLocation( "Change Prompt", 141, OFS_ENDMSG, 135, 269, 326);
ofsSetWindowLocation( "Save Folder", 145, OFS_ENDMSG, 135, 269, 307);
ofsSetWindowLocation( "Destination Details", 149, OFS_ENDMSG, 135, 269, 307);
ofsSetWindowLocation( "Folder Query", 153, OFS_ENDMSG, 135, 240, 216);
ofsSetWindowLocation( "Autosize All", 157, OFS_ENDMSG, 135, 307, 269);
ofsSetWindowLocation( "Calendar", 197, OFS_ENDMSG, 135, 96, 96);
ofsSetWindowLocation( "Progress", 201, OFS_ENDMSG, 135, 288, 336);
ofsSendRecv(2); //ClientSeqNo=8|MsgCount=10|1128450789.296


DO_SLEEP(2);
ofsRemoveFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsFocus( "BUTTON", 293, OFS_ENDMSG, 174 );
```

```
ofsDeActivateWindow( "Material Workbench", 161, OFS_ENDMSG, 247 );
ofsSendRecv(2); //ClientSeqNo=9|MsgCount=3|1128450790.734


DO_SLEEP(1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 1, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 2, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 3, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 4, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 5, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 6, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 7, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 8, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 9, 3);
ofsSendRecv(1); //ClientSeqNo=10|MsgCount=9|1128450793.250


DO_SLEEP(3);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 1, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 2, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 3, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 4, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 5, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 6, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 7, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 8, 1);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1600, 1200);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 9, 3);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 1, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 2, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 3, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 4, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 5, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 6, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 7, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 8, 1);
ofsLOVRequestRow( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 451, 9, 3);
ofsSendRecv(1); //ClientSeqNo=11|MsgCount=19|1128450795.187

ofsSendRecv(1); //ClientSeqNo=12|MsgCount=0|1128450795.187
```

```
DO_SLEEP(2);

DO_SLEEP(2);
ofsActivateWindow( "Material Workbench", 161, OFS_ENDMSG, 247 );
ofsSetPropertyVoid( "LISTVALUESDIALOG", 615, OFS_ENDMSG, 105 );
ofsSendRecv(1); //ClientSeqNo=13|MsgCount=2|1128450801.593


DO_SLEEP(4);
ofsSendRecv(2); //ClientSeqNo=14|MsgCount=0|1128450802.921


DO_SLEEP(1);
ofsSetCursorPosition( "TEXTFIELD", 89, OFS_ENDMSG, 193, "0" );
ofsFocus( "TABCONTROL", 255, OFS_ENDMSG, 174 );
ofsFocus( "BUTTON", 73, OFS_ENDMSG, 174 );
ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsRemoveFocus( "BUTTON", 73, OFS_ENDMSG, 174 );
ofsFocus( "TEXTFIELD", 89, OFS_ENDMSG, 174 );
ofsSendRecv(2); //ClientSeqNo=15|MsgCount=6|1128450804.515


DO_SLEEP(2);
ofsSendRecv(2); //ClientSeqNo=16|MsgCount=0|1128450805.828


DO_SLEEP(1);
ofsCloseWindow( "NAVIGATOR", 28, OFS_ENDMSG, 216 );
ofsSendRecv(1); //ClientSeqNo=17|MsgCount=1|1128450808.656


DO_SLEEP(3);
ofsDeActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsSetPropertyInteger( "CUSTOMCONTROL", 616, OFS_ADD, 1, "20002" );
ofsSetPropertyInteger( "CUSTOMCONTROL", 616, OFS_ENDMSG, 1, "0" );
ofsSendRecv(1); //ClientSeqNo=18|MsgCount=2|1128450810.875


DO_SLEEP(2);
ofsSetCursorPosition( "TEXTFIELD", 89, OFS_ENDMSG, 193, "0" );
```

```
  ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
  ofsSendRecv(2); //ClientSeqNo=19|MsgCount=2|1128450812.406


  DO_SLEEP(2);
  ofsSetPropertyInteger( "FORMWINDOW", 6, OFS_ENDMSG, 130, "1" );
  ofsSendRecv(3); //ClientSeqNo=20|MsgCount=1|1128450813.609


  DO_SLEEP(1);
  ofsHTTPDisconnect();

  DO_SetTransactionCleanup();
  /* Clear up some internal storage used for DO_SetValue() */
  DO_HttpCleanup();
  for(i=0; i<3; i++)
  {
     free(Field[i]);
     Field[i]=NULL;
  }
  for(i=0; i<2; i++)
  {
     free(Anchor[i]);
     Anchor[i]=NULL;
  }
  for(i=0; i<1; i++)
  {
     free(ActionURL[i]);
     ActionURL[i]=NULL;
  }
//----------------------------------------------------------------------------
// Free up memory that was allocated earlier
//----------------------------------------------------------------------------
  if (NULL != sessionid) free(sessionid);
  if (NULL != nvis510) free(nvis510);
  if (NULL != cookieHeaderValue) free(cookieHeaderValue);
  if (NULL != icx_ticket) free(icx_ticket);

  END_TRANSACTION();
  DO_FreeHttp();
```

```
  EXIT();
  return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");
    DO_FreeHttp();

    EXIT();
}
```

SAP Scripts

SAP Script Samples

You can address specific situations or resolve certain problems by modifying converted SAP scripts. The samples shown here include a description of the problem, the procedure for implementing the modification, and samples of a modified script. Script modifications are discussed for:

Checking and Handling SAPGuiCheckScreen Errors

Overview: Handling SAPGuiCheckScreen Errors

The SAP screen returned from the SAP server during playback can differ from the screen status of the session during record. The SAPGuiCheckScreen call returns a Boolean value that indicates whether the screen status is expected or unexpected. By checking the status returned from this call, you can build more extensive error handling capability into the script.

Create custom error handling code by placing the SAPGuiCheckScreen call within an `If` conditional that checks the return value, then writing custom error handling code for the functions that fail within that conditional block.

The sample script shows the code for creating error code blocks to use with checking the status of a returned SAP screen.

Sample: SAP Script with SAPGuiCheckScreen Error Handler

The following shows an SAP script snippet with SAPGuiCheckScreen Error Handler. Points of interest in the script are highlighted in bold.

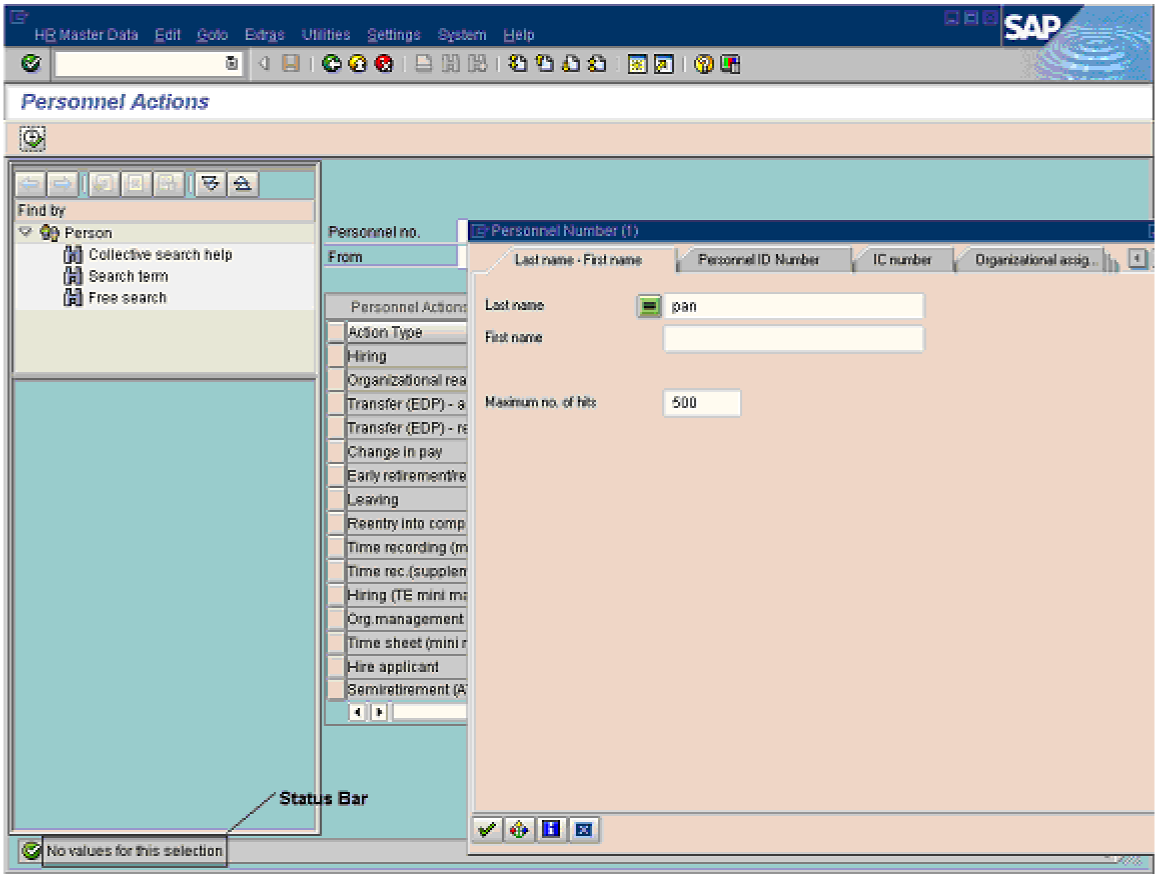
Sample Script Snippet

```
…beginning of script…

SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:0200/subSA_200_1:SAPLEXAMP
LE_ENTRY_SCREEN:0800/cntlCC_HTML_INDEX/shellcont/shell");
SAPGuiCmd3(GuiCtrlHTMLViewer, SapEvent, "","","sapevent:FLD_SHORT?FLD" );

/// In this small example, if any of the following are FALSE
///    OKCode == "BIBS"
///    screen name == "SAPLEXAMPLE_ENTRY_SCREEN"
///   title == "Style Guide: Dropdown list"
/// …then the block of error code is processed!

   BOOL bRetSts = SAPGuiCheckScreen (
          "BIBS",
          "SAPLEXAMPLE_ENTRY_SCREEN",
          "Style Guide: Dropdown list" );

   if (bRetSts != TRUE)
   {
      RR__printf("Style Guide: Dropdown list not found");
      SAPGui_error_handler(s_info, "Error: Style Guide: Dropdown list");
   }

   DO_SLEEP(9);

   SAPGuiCmd3(GuiCtrlHTMLViewer, SapEvent, "","","sapevent:PUB_SHORT?PUB" );

…end of script…
```

Checking the SAP Status Bar

Overview: Checking the Status Bar

The SAP status bar displays error and status messages. When running an SAP script, you can check the status bar to determine whether the script is executing properly.



Conclusion

Use the SAPGuiCheckStatusbar command to test for certain status responses in the SAP environment and take actions based on messages returned from the SAP server. The sample script illustrates this procedure.

Sample Script: Checking the SAP Status Bar

The SAPGuiCheckStatusbar command is used in the following script example to test for certain status responses. The code added to check the status bar is shown in bold.

Sample Script

```
...
SAPGuiPropIdStr("wnd[0]");//1109615021.466
SAPGuiCmd1(GuiMainWindow,SendVKey,4);
SAPGuiCheckScreen("PA40","SAPMP50A","Personnel Actions");  //1109615021.481

DO_SLEEP(15);

SAPGuiPropIdStr(""
   "wnd[1]/usr/tabsG_SELONETABSTRIP/tabpTAB001/ssubSUBSCR_PRESEL:SAPLSDH4:0220/sub:"
"SAPLSDH4:0220/txtG_SELFLD_TAB-LOW[0,24]" ); //1109615036.231
SAPGuiCmd1(GuiTextField,PutText,"pan");
SAPGuiCmd1(GuiTextField,PutCaretPosition,3);

SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");//1109615036.246
SAPGuiCmd0(GuiButton,Press);
SAPGuiCheckScreen("PA40","SAPLSDH4","Restrict Value Range"); //1109615036.246

// Check to determine if the name we chose is found in db
```

```
// If not stop the script should not continue
BOOL bRetSts =
SAPGuiCheckStatusbar("wnd[0]/sbar", "No values for this selection");
if (bRetSts)
{
    RR__printf(" No such last name in Database");
    SAPGui_error_handler(s_info," End Now No such name in Database");
}
...
```

Handling Multiple SAP Logons in a Single Script

Modifying the Script for Multiple Logons

Load testing requirements for SAP may include allowing for multiple logons in a script session. When you incorporate multiple logon functionality for an SAP script, you must add a code block to the script to see if the multiple logon option is available, and whether the user has selected the radio button that allows concurrent connections from the same user.

To modify a script to handle multiple logons:

1. Find the following line of code in a script you just captured.

Tip: This line of code should be near the top of the script after the logon commands.

```
SAPGuiCheckScreen("S000","SAPMSYST","SAP");
```

2. After this line of code, add the following code block. This shows whether the multiple logon option is available and if the user has pressed the Allow Concurrent Connections from the Same User radio button on the SAP logon screen.

```
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");

DO_SLEEP(20);

SAPGuiCmd0(GuiRadioButton,Select);

SAPGuiCmd0(GuiRadioButton,SetFocus);

SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");

SAPGuiCmd0(GuiButton,Press);

SAPGuiCheckScreen("S000","SAPMSYST","License Information for Multiple Logon");

SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");

SAPGuiPropIdStr("wnd[0]"); // sets focus back to wind[0]
```

This code acts like an If conditional that is not in a control loop. It can be stated as follows:

- IF the multi-logon screen appears, select radio button option 2. In this code snippet, radio button 2 tells the server to allow concurrent connections from the same user.
- THEN check to make certain the script is synced to the "License Information for Multiple Logon" section.
- ELSE do nothing and continue with the script.

Conclusion

This technique enables support for multiple SAP logon sessions in a script. It asks SAP whether the multi-logon radio button is selected and, if it is, simulates selecting the button during replay. This allows you to run the same script with many virtual users under a single SAP logon ID.

The sample original script and sample modified script illustrate this process.

Sample: Original SAP Script for Multiple Logons

The following sample shows an original script converted from capture. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  multilogon.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
 *      - SAP
 */

/* Converted using the following options:
 * General:
 *  Line Split                          : 132 characters
 *  Sleep Seconds                       : 1
 *  Constants to Variables              : Yes
 *  Remove Quotes                       : No
 *  Tabs To Spaces                      : No
 *  Auto Checkpoints                    : No
 * SAP
 *  Version                             : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"

extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("capture.cpp");
```

```
HRESULT hr = CoInitialize(0);

if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

SAPGuiSetCheckScreenWildcard('*');

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

try{

     SAPGuiConnect( s_info,"testsap620");

     SAPGuiApplication(RegisterROT);

     SAPGuiVerCheckStr("6204.119.32");

     //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
     //SapApplication.OpenConnection ("testsap620")
     //Set Session    = SapApplication.Children(0).Children(0)

     DO_SLEEP(18);

     SAPGuiPropIdStr("wnd[0]");//1057828784.513
     SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

     DO_SLEEP(16);

     SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
     SAPGuiCmd1(GuiTextField,PutText,"qaload1");

     SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
     SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
     SAPGuiCmd0(GuiPasswordField,SetFocus);
     SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

     SAPGuiPropIdStr("wnd[0]");//1057828800.836
     SAPGuiCmd1(GuiMainWindow,SendVKey,0);
     SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856

     DO_SLEEP(6);
     SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

     DO_SLEEP(3);

     SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
     SAPGuiCmd0(GuiButton,Press);

/// This is the command to look for. The parameters indicate that this
/// SAP Easy Access session screen.

     SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
     Access");//1057828809.859

     DO_SLEEP(2);

     SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
     SAPGuiCmd0(GuiButton,Press);
     SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402

     } // end try
```

```
        catch (_com_error e){
        char buffer[1024];
        sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
        *)e.Description(), S_task_id);
        SAPGui_error_handler(s_info,buffer);
        } // end catch

   DO_SetTransactionCleanup();

   SAPGuiApplication(RevokeROT);

   END_TRANSACTION();

   REPORT(SUCCESS);
   EXIT();
   return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");
   EXIT();
}
```

Sample: Modified SAP Script for Multiple Logons

The following sample shows a SAP script modified to handle multiple logon sessions. Added code snippet and related comments are highlighted in bold.

Sample Script

```
/*
 *  multilogon.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
 *      - SAP
 */
/* Converted using the following options:
 * General:
 *  Line Split                        : 132 characters
 *  Sleep Seconds                     : 1
 *  Constants to Variables            : Yes
 *  Remove Quotes                     : No
 *  Tabs To Spaces                    : No
 *  Auto Checkpoints                  : No
 * SAP
 *  Version                           : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"
```

```
extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("multilogon.cpp");

   HRESULT hr = CoInitialize(0);

   if( hr != ERROR_SUCCESS )
       RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

   SAPGuiSetCheckScreenWildcard('*');

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   try{

        SAPGuiConnect( s_info,"testsap620");

        SAPGuiApplication(RegisterROT);

        SAPGuiVerCheckStr("6204.119.32");

        //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
        //SapApplication.OpenConnection ("testsap620")
        //Set Session    = SapApplication.Children(0).Children(0)

        DO_SLEEP(18);

        SAPGuiPropIdStr("wnd[0]");//1057828784.513
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(16);

        SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
        SAPGuiCmd1(GuiTextField,PutText,"qaload1");

        SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
        SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
        SAPGuiCmd0(GuiPasswordField,SetFocus);
        SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

        SAPGuiPropIdStr("wnd[0]");//1057828800.836
        SAPGuiCmd1(GuiMainWindow,SendVKey,0);

/// Scripting: Step 1
/// Here is the command to look for in the script.  Note the parameters
/// of the call, they are consistent in every SAP script.

        SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856
```

```
/// Scripting: Step 2
/// Add this code snippet in bold after the SAPGuiCheckScreen which
/// confirms the login screen is present.

        SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");

               DO_SLEEP(20);
               SAPGuiCmd0(GuiRadioButton,Select);
               SAPGuiCmd0(GuiRadioButton,SetFocus);
               SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");
               SAPGuiCmd0(GuiButton,Press);
               SAPGuiCheckScreen("S000","SAPMSYST","License Information for Multiple
               Logon");

        SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");

        SAPGuiPropIdStr("wnd[0]"); // sets focus back to wind[0]

        DO_SLEEP(6);
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(3);

        SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
        Access");//1057828809.859

        DO_SLEEP(2);

        SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402

        } // end try

        catch (_com_error e){
        char buffer[1024];
        sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
        *)e.Description(), S_task_id);
        SAPGui_error_handler(s_info,buffer);
        } // end catch

    DO_SetTransactionCleanup();

    SAPGuiApplication(RevokeROT);

    END_TRANSACTION();

    REPORT(SUCCESS);
    EXIT();
    return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");
    EXIT();
}
```

Required SAP Commands to Support Transaction Restart

Overview: Transaction Restart Commands

If your load testing requirements include supporting transaction restarts, review the following commands and ensure that those required to support transaction restart are present.

Note: These are automatically added during the script conversion process.

Scripting

Required commands and associated code statements for the SAP script are:

- SAPGuiApplication(RegisterROT) - This command sets or resets the SAP runtime object library (ROT). Executing this command ensures the SAP environment is in an initial state at the beginning of the script.
- SAPGuiApplication(RevokeROT) - This command clears the SAP runtime object library (ROT). Executing this command ensures the memory SAP environment is de-allocated.
- SAPGui_error_handler(s_info, buffer) - When scripting an error code block, be sure to use the SAPGui_error_handler command instead of the RR__FailedMsg command if the Transaction Restart option is used.

Note: These functions are required, but additional SAP script commands are also essential to run an SAP script.

Conclusion

If unexpected playback errors occur when playing back an edited script with Transaction Restart enabled, perform a review of the commands to ensure that those required to support transaction restart are present.

The sample script shows the commands used in the script.

Sample: Required SAP Commands Supporting Transaction Restart

The following sample script shows the SAP commands required to support transaction restart. Essential Transaction Restart commands are highlighted in bold.

Sample Script

```
/*
 *  capturewithrestart.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
 *      - SAP
 */

/* Converted using the following options:
 * General:
 *  Line Split                       : 132 characters
 *  Sleep Seconds                    : 1
 *  Constants to Variables           : Yes
 *  Remove Quotes                    : No
 *  Tabs To Spaces                   : No
 *  Auto Checkpoints                 : No
 * SAP
 *  Version                          : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
```

```
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"

extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)

{

   /* Declare Variables */

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("capturewithrestart.cpp");

   HRESULT hr = CoInitialize(0);

   if( hr != ERROR_SUCCESS )
       RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

   SAPGuiSetCheckScreenWildcard('*');

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   try{

        SAPGuiConnect( s_info,"testsap620");

/// This command registers the SAP Runtime Object Table (ROT).
/// Setting the ROT effectively sets or resets the SAP runtime environment.

        SAPGuiApplication(RegisterROT);

        SAPGuiVerCheckStr("6204.119.32");

        //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
        //SapApplication.OpenConnection ("testsap620")
        //Set Session    = SapApplication.Children(0).Children(0)

        DO_SLEEP(18);

        SAPGuiPropIdStr("wnd[0]");//1057828784.513
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(16);

        SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
        SAPGuiCmd1(GuiTextField,PutText,"qaload1");
```

```
        SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
        SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
        SAPGuiCmd0(GuiPasswordField,SetFocus);
        SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

        SAPGuiPropIdStr("wnd[0]");//1057828800.836
        SAPGuiCmd1(GuiMainWindow,SendVKey,0);
        SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856


        DO_SLEEP(6);
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(3);

        SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
        Access");//1057828809.859

        // Check to determine if the name we chose is found in db
        // If not stop the script should not continue

        BOOL bRetSts =
        SAPGuiCheckStatusbar("wnd[0]/sbar", "No values for this selection");

        if (bRetSts)
        {
            RR__printf(" No such last name in Database");
/// Use the SAPGui_error_handler call within code blocks that process
/// error conditions.  Do not use the RR__FailedMsg call in scripts
/// where the Transaction Restart option will be enabled!

            SAPGui_error_handler(s_info," End Now No such name in Database");
        }

        DO_SLEEP(2);

        SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402


        } // end try

        catch (_com_error e){
        char buffer[1024];
        sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
        *)e.Description(), S_task_id);
        SAPGui_error_handler(s_info,buffer);
        } // end catch

    DO_SetTransactionCleanup();

/// This command clears the SAP Runtime Object Table (ROT).
/// Revoking the ROT deletes the SAP runtime environment from memory.

    SAPGuiApplication(RevokeROT);

    END_TRANSACTION();

    REPORT(SUCCESS);
    EXIT();
    return(0);
}
```

```
void abort_function(PLAYER_INFO *s_info)
{
     RR__printf("Virtual User ABORTED.");
     EXIT();
}
```

Required SAP Script Commands

Overview: Required SAP Script Commands

Certain commands must be present in an SAP script. These commands are created automatically during the conversion process. Most of the commands are before the BEGIN_TRANSACTION statement. Review this section if you are having unexpected issues after script editing.

> Note: If the SAP script supports transaction restarting, review Required Commands for Supporting Transaction Restart in SAP.

Required Script Commands

Required commands and associated code statements for the SAP script are:

SET_ABORT_FUNCTION(abort function);

DEFINE_TRANS_TYPE("capture.cpp");

HRESULT hr = CoInitialize(0);

if(hr != ERROR_SUCCESS)

RR__FailedMsg(s_info, "ERROR initializing COM");

SAPGuiSetCheckScreenWildcard(‘*’);

SYNCHRONIZE();

> Note: These functions are required, but additional SAP API script commands are also essential to run an SAP script.

Conclusion

When encountering unexpected compiler errors after script editing, review the script to ensure all required commands are present. This might reveal a problem created by the script edits, especially when moving the transaction loop. You must take care when doing extensive script editing not to accidentally remove critical commands. Recording the transaction again and doing a windiff comparison can also help when unexpected compiler errors occur.

The sample script shows the statements used in the script.

Sample: SAP Script with Commands

The following sample is a SAP Script with required commands. Required commands in the script are highlighted in bold.

Sample Script

```
/*
 *  capture.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
```

```
 *      - SAP
 */

/* Converted using the following options:
 * General:
 *  Line Split                     : 132 characters
 *  Sleep Seconds                  : 1
 *  Constants to Variables         : Yes
 *  Remove Quotes                  : No
 *  Tabs To Spaces                 : No
 *  Auto Checkpoints               : No
 * SAP
 *  Version                        : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"

extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{
   /* Declare Variables */

/// These script functions in bold must be present and
/// before the SYNCHRONIZE command in every SAP script.

   ACTION();

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("capture.cpp");

   HRESULT hr = CoInitialize(0);

   if( hr != ERROR_SUCCESS )

       RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

   SAPGuiSetCheckScreenWildcard('*');

   SYNCHRONIZE();

   BEGIN_TRANSDO_SetTransactionStart();

   try{
```

```
        SAPGuiConnect( s_info,"testsap620");

        SAPGuiApplication(RegisterROT);

        SAPGuiVerCheckStr("6204.119.32");

        //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
        //SapApplication.OpenConnection ("testsap620")
        //Set Session    = SapApplication.Children(0).Children(0)

        DO_SLEEP(18);

        SAPGuiPropIdStr("wnd[0]");//1057828784.513
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(16);

        SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
        SAPGuiCmd1(GuiTextField,PutText,"qaload1");

        SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
        SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
        SAPGuiCmd0(GuiPasswordField,SetFocus);
        SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

        SAPGuiPropIdStr("wnd[0]");//1057828800.836
        SAPGuiCmd1(GuiMainWindow,SendVKey,0);
        SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856

        DO_SLEEP(6);
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(3);

        SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
        Access");//1057828809.859

        DO_SLEEP(2);

        SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402

        } // end try

        catch (_com_error e){
        char buffer[1024];
        sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
        *)e.Description(), S_task_id);
        SAPGui_error_handler(s_info,buffer);
        } // end catch

    DO_SetTransactionCleanup();

    SAPGuiApplication(RevokeROT);

    END_TRANSACTION();

    REPORT(SUCCESS);
    EXIT();
    return(0);
}
```

```
void abort_function(PLAYER_INFO *s_info)
{
   RR__printf("Virtual User ABORTED.");
   EXIT();
}
```

Retrieving SAP Counter Data

Modifying the Script to Retrieve SAP Counter Data

SAP scripts can retrieve customer counter information for each virtual user. By inserting code snippets that use the SAPGui scripting API into the SAP script, you can obtain and save SAP server information.

To modify the script to retrieve SAP counter data:

1. Declare and initialize the counter identification (ID) variables using the **int** data type. You should declare a variable for each counter value to be extracted. The DEFINE_COUNTER macro initializes the declared counter identifier variable and creates a holder for the value in the timing file.
2. Declare and initialize the variable to hold the actual SAP counter value. You should declare the variable using a datatype that can hold any expected value for the counter. Usually a long is appropriate.
3. Retrieve the counter information from the SAP server using the SAPGuiSessionInfo command. The value is placed in the variable you created in Step 2. The first parameter is the SAP property object corresponding to the counter. The second parameter is the variable to hold the value.
4. Save the counter value to the timing file. The COUNTER_VALUE macro command extracts the value from the server. The value is extracted to the variable created in Step 2. It is stored in the timing file using the associated ID created in Step 1.

Conclusion

Following these techniques, you can obtain customer counter information from the SAP server, save it to the virtual user's timing file, and view it in Analyze. The sample original script and sample modified script illustrate this modification.

Sample: Original SAP Script with Counters

The following sample shows an original SAP script extract with counters. Points of interest in the script are highlighted in bold.

Sample Script

```
/*
 *  counters.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
 *      - SAP
 */

/* Converted using the following options:
 * General:
 *  Line Split                          : 132 characters
 *  Sleep Seconds                       : 1
 *  Constants to Variables              : Yes
 *  Remove Quotes                       : No
 *  Tabs To Spaces                      : No
```

```
 *  Auto Checkpoints                 : No
 * SAP
 *  Version                          : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"

extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{

   /* Declare Variables */

/// Declare the SAP custom counter variable IDs
/// Also declare any SAP counter value variables.

   SET_ABORT_FUNCTION(abort_function);

   DEFINE_TRANS_TYPE("counters.cpp");

   HRESULT hr = CoInitialize(0);

   if( hr != ERROR_SUCCESS )
       RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

   SAPGuiSetCheckScreenWildcard('*');

/// Initialize the SAP custom counter variable IDs.

   SYNCHRONIZE();

   BEGIN_TRANSACTION();

   DO_SetTransactionStart();

   try{

        SAPGuiConnect( s_info,"testsap620");

        SAPGuiApplication(RegisterROT);

        SAPGuiVerCheckStr("6204.119.32");

        //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
        //SapApplication.OpenConnection ("testsap620")
```

```
        //Set Session    = SapApplication.Children(0).Children(0)

        DO_SLEEP(18);

        SAPGuiPropIdStr("wnd[0]");//1057828784.513
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(16);

        SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
        SAPGuiCmd1(GuiTextField,PutText,"qaload1");

        SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
        SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
        SAPGuiCmd0(GuiPasswordField,SetFocus);
        SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);

        SAPGuiPropIdStr("wnd[0]");//1057828800.836
        SAPGuiCmd1(GuiMainWindow,SendVKey,0);
        SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856

        DO_SLEEP(6);
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(3);

        SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
        Access");//1057828809.859

        DO_SLEEP(2);

        SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
        SAPGuiCmd0(GuiButton,Press);

/// This is where we would like to retrieve the RoundTrips
/// and Flushes counter.  Here the SAPGuiSessionInfo command
/// will be inserted to retrieve these SAP counter values from the server.

        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402

/// Here is where the counter information will actually be
/// written to the timing file.

    } // end try

    catch (_com_error e){
        char buffer[1024];
        sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
        *)e.Description(), S_task_id);
        SAPGui_error_handler(s_info,buffer);
    } // end catch

    DO_SetTransactionCleanup();

    SAPGuiApplication(RevokeROT);

    END_TRANSACTION();

    REPORT(SUCCESS);
    EXIT();
    return(0);
}

void abort_function(PLAYER_INFO *s_info)
```

```
{
   RR__printf("Virtual User ABORTED.");
   EXIT();
}
```

Sample: Modified SAP Script with Custom Counters

The following sample shows a modified SAP script with SAP custom counters. Changes to Original script are highlighted in bold.

Sample Script

```
/*
 *  counters.cpp
 *
 *  Script Converted on July 20, 2004 at 08:43:23 AM
 *  Generated by Compuware QALoad convert module version 5.2.0 build 73
 *
 *  This script contains support for the following middlewares:
 *      - SAP
 */

/* Converted using the following options:
 * General:
 *  Line Split                     : 132 characters
 *  Sleep Seconds                  : 1
 *  Constants to Variables         : Yes
 *  Remove Quotes                  : No
 *  Tabs To Spaces                 : No
 *  Auto Checkpoints               : No
 * SAP
 *  Version                        : 6204.119.32
 */

#define SCRIPT_VER 0x00000205UL

#include <stdio.h>
#include <windows.h>
#include <atlbase.h>
#include <objbase.h>
#include "do_SAPCCOM.h"
#include <atlwin.h>
#include <atlcom.h>
#include <atlhost.h>
#include "cscript.h"
#include "do_SapGui.h"
#include "mwCommon.h"

extern "C" {
#include "smacro.h"
}

/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);

#ifndef NULL
#define NULL  0
#endif

extern "C" int rhobot_script(PLAYER_INFO *s_info)
{

   /* Declare Variables */
```

```
/// Scripting: Step 1

    int id1, id2, id3, id4;
    long lRoundTrips,lFlushes;

    SET_ABORT_FUNCTION(abort_function);

    DEFINE_TRANS_TYPE("counters.cpp");

    HRESULT hr = CoInitialize(0);

    if( hr != ERROR_SUCCESS )
        RR__FailedMsg(s_info,"SAP: ERROR initializing COM");

    SAPGuiSetCheckScreenWildcard('*');

/// Scripting: Step 2

    // "Counter Group", "Counter Name", "Counter Units
    // (Optional)", Data Type, Counter Type.

    id1 = DEFINE_COUNTER("Cumulative Group", "Cumulative RoundTrips", 0, DATA_LONG,
    COUNTER_CUMULATIVE);

    id2 = DEFINE_COUNTER("Cumulative Group", "Cumulative Flushes", 0, DATA_LONG,
    COUNTER_CUMULATIVE);

    id3 = DEFINE_COUNTER("Instance Group", "Instance RoundTrips", 0, DATA_LONG,
    COUNTER_INSTANCE);

    id4 = DEFINE_COUNTER("Instance Group", "Instance Flushes", 0, DATA_LONG,
    COUNTER_INSTANCE);


    SYNCHRONIZE();

    BEGIN_TRANSACTION();

    DO_SetTransactionStart();

    try{

         SAPGuiConnect( s_info,"testsap620");

         SAPGuiApplication(RegisterROT);

         SAPGuiVerCheckStr("6204.119.32");

         //Set SapApplication = CreateObject("Sapgui.ScriptingCtrl.1")
         //SapApplication.OpenConnection ("testsap620")
         //Set Session    = SapApplication.Children(0).Children(0)

         DO_SLEEP(18);

         SAPGuiPropIdStr("wnd[0]");//1057828784.513
         SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

         DO_SLEEP(16);

         SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");//1057828800.786
         SAPGuiCmd1(GuiTextField,PutText,"qaload1");

         SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");//1057828800.796
         SAPGuiCmd1Pwd(GuiPasswordField,PutText,"~encr~0000x_'9d");
         SAPGuiCmd0(GuiPasswordField,SetFocus);
         SAPGuiCmd1(GuiPasswordField,PutCaretPosition,3);
```

```
        SAPGuiPropIdStr("wnd[0]");//1057828800.836
        SAPGuiCmd1(GuiMainWindow,SendVKey,0);
        SAPGuiCheckScreen("S000","SAPMSYST","SAP");//1057828800.856

        DO_SLEEP(6);
        SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,92,34,false);

        DO_SLEEP(3);

        SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[15]");//1057828809.839
        SAPGuiCmd0(GuiButton,Press);
        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy
        Access");//1057828809.859


        DO_SLEEP(2);

        SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");//1057828811.382
        SAPGuiCmd0(GuiButton,Press);

/// Scripting: Step 3

        SAPGuiSessionInfo(GetRoundTrips,lRoundTrips);
        SAPGuiSessionInfo(GetFlushes,lFlushes);

        SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");//1057828811.402

/// Scripting: Step 4

        COUNTER_VALUE( id1,lRoundTrips);
        COUNTER_VALUE( id2,lFlushes);
        COUNTER_VALUE( id3,lRoundTrips);
        COUNTER_VALUE( id4,lFlushes);

    } // end try

    catch (_com_error e){
       char buffer[1024];
       sprintf(buffer,"SAP: EXCEPTION 0x%x  %s for VU(%i)\n",e.Error(), (char
       *)e.Description(), S_task_id);
       SAPGui_error_handler(s_info,buffer);
    } // end catch

    DO_SetTransactionCleanup();

    SAPGuiApplication(RevokeROT);

    END_TRANSACTION();
    REPORT(SUCCESS);
    EXIT();
    return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User ABORTED.");
    EXIT();
}
```

Winsock Scripts

Winsock Script Samples

You can address specific situations or resolve certain problems by modifying converted Winsock scripts. These samples show a description of the problem, the procedure for implementing the modification, and samples of a modified script. Script modifications are discussed for:

Handling Winsock Connection Problems

Accessing Local and Remote Network Addresses

Using Central Datapools within a Winsock Script

Using Local Datapools within a Winsock Script

Accessing Server Replies with DO_WSK_Read

Accessing Server Replies with DO_WSK_Recv

Receiving Winsock UDP Data with DO_WSK_Recvfrom

Sending Variable Data with DO_WSK_Send

Sending Variable Data using DO_WSK_SendAll

Sending Variable Data using DO_WSK_SendTo

Sending Variable Data using DO_WSK_Write

Accessing Server Replies Using Response() and ResponseLength()

Parsing Server Replies Using ScanFloat and ScanInt

Parsing Server Replies Using ScanSkip and ScanString

Parsing Server Replies Using SkipExpr and ScanSkip

Handling Winsock Connection Problems

Overview: Handling Winsock Connection Problems

When a remote application is handling other tasks and is not able to accept a connection, you can modify your script so that it retries the connection. When implementing this sort of functionality, be sure you:

- Limit the number of connection retries so that the script doesn't execute indefinitely.
- Turn off the abort flag before the connection and turn it back on immediately after the connection loop. Otherwise the virtual user aborts when a connection fails.

Conclusion

The sample script illustrates how to create a script that attempts to reconnect to a remote application if the first connection fails.

Caution: It is very import that you specify a limit to the number of connection retries or the script runs in an endless loop. This may create a denial of service attack on the remote application.

Sample: Handling Winsock Connection Problems

The following script snippet shows an example of how to handle connection problems. The code in bold shows how you can limit the retires and turn off the abort flag.

Sample Script Snippet

```
…
/* Declare Variables */
//Declared nNumRetries to keep track of the number of retry attempts that are made
int nNumRetries = 0;
//Declared "ret" to store the return code from DO_WSK_Connect()
int ret = -1;
…
BEGIN_TRANSACTION();

DO_WSK_Socket(S2, AF_INET, SOCK_STREAM, IPPROTO_TCP);

//In order to handle the failure for the DO_WSK_Connect the script needs to
// turn off the abort on error flag. After the connection is made the flag
// should be turned back on.
s_info->bAbortOnError = 0;

//The example below will try to make a connection to the remote application,
// and if it fails it will repeat the attempt up to 5 times.
while( ret = -1 && nNumRetries < 5)
{
   //Attempts to connect to the remote application.
   ret = DO_WSK_Connect(S2, "10.4.26.24", 123, AF_INET);

   if(ret == -1)
   {
        //pause between connection attempts
        DO_MSLEEP(500);
        //increment the counter for the number of retry attempts.
        nNumRetries++;
   }
}

//turning the abort flag back on.
s_info->bAbortOnError = 1;
//If the max number of retries has been met then this virtual user should fail.
if(nNumRetries == 5)
RR__FailedMsg(s_info,"ERROR: Max number of connection retries has been reached");
...

DO_WSK_Closesocket(S2);
```

Parsing Server Replies Using SkipExpr and ScanSkip

Overview: Parsing Server Replies

You can find a string within the reply returned from the sever using SkipExpr() and ScanExpr(). SkipExpr() searches the internal buffer that contains the response received in the DO_WSK_Expect() for the first occurance of the string. It places the pointer in the position immediately following the string. Then use ScanExpr() to search for the end of the string. This saves the buffer following the first occurrence of the string used in SkipExpr(), up to and including the string in ScanExpr(). The first parameter of ScanExpr() takes a Unix-style regular expression. The following table shows some of the most common expressions.

Character	Meaning
"."	Matches the end of a string
"*"	Matches any number of characters

“?”	Matches any one character

Conclusion

Use the SkipExpr() and ScanExpr() to find a string returned from the server. The sample script illustrates this technique.

Sample: Parsing Server Replies Using SkipExpr and ScanSkip

In this example, the buffer returned from the server is: "sessionid=1234567890abc”, and you are retrieving everything after the “=”, up to and including “abc”. The message “length = 13 string = 1234567890abc” is printed to the player buffer window. The code added to check the status bar is shown in bold.

Sample Script

```
/* Declare Variables */
char temp[35];
int size = 0;
…
BEGIN_TRANSACTION();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);

DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);
//Below indicates the buffer that was received at capture time.
/* 23 bytes: sessionid=1234567890abc */
DO_WSK_Expect(S1, "c");

//Finds the string “sessionid=” within the buffer that was received, and
//the moves the pointer to the next position within the received buffer.
SkipExpr("sessionid=");

//Copies any data from the current buffer position up to the string “abc”
//into the variable temp.
size=ScanExpr(".*abc" , temp);

//prints the string that was found to the playerbuffer window.
RR__printf("length =  %d string  = %s", size, temp);

DO_WSK_Closesocket(S1);
```

Accessing Local and Remote Network Addresses

You can retrieve the IP address or port to which a socket handle is connected, or retrieve the IP address and port to which a socket handle is bound. The sample script illustrates how to retrieve and store socket address and port information. The required code is shown in bold.

Sample Script

```
...
/* Declare Variables */

//Belows are two socket structs that will store the address and port information.
struct sockaddr_in RemoteAddr;
struct sockaddr_in LocalAddr;
```

```
…
BEGIN_TRANSACTION();
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_TCP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
/* Socket S2 was bound to address 0.0.0.0 on port 3320 (hibyte=12,lobyte=248) */

//Called after the DO_WSK_Bind() to see which address the socket was bound to
LocalAddr.sin_addr.s_addr = ntohl(GetLocalAddr(S1));

//Called after the DO_WSK_Bind() to see which port the socket was bound to
LocalAddr.sin_port  = GetLocalPort(S1);

DO_WSK_Connect(S1, "10.4.26.24", 80, AF_INET);

//Called to retrieve the remote address that S1 is connected to.
RemoteAddr.sin_addr.s_addr = ntohl(GetRemoteAddr(S1));

//Called to retrieve the remote port that S1 is connected to.
RemoteAddr.sin_port = ntohs(GetRemotePort(S1));

//The function below will print the remote and local address and port information
// to the playerbuffer. Within the RR__printf() the script is using the socket function
// "inet_ntoa()" to convert the IP from an unsigned long to a string
// format, "XXX.XXX.XXX.XXX".
RR__printf("Remote: address=%s port=%d",inet_ntoa(RemoteAddr.sin_addr),
RemoteAddr.sin_port);
RR__printf("Local: address=%s port=%d",inet_ntoa(LocalAddr.sin_addr), LocalAddr.sin_port);

DO_WSK_Closesocket(S1);

…
```

Conclusion

When this code is executed, a message is printed to the playerbuffer, such as:

```
VU  0 : Remote: address=10.4.26.24 port=80

VU 0 : Local: address=10.15.16.26 port=1125
```

Accessing Server Replies Using Response and ResponseLength

You can save the entire reply that a server returns using Response() and ResponseLength(). When you call Response() directly after the DO_WSK_Expect(), it returns a pointer to the data received by the DO_WSK_Expect(). To receive the length of the received reply, call the ResponseLength(). This returns the number of characters received.

The sample script illustrates how to use these commands to save a server reply. In this sample, the code in bold shows how to use Response() and ResponseLength().

Sample Script

```
/* Declare Variables */
//Below are variables needed for this example
int x = 0;
char *temp;
…
BEGIN_TRANSACTION();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
```

```
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//Below is the actual buffer that was returned at capture time.
/* 21 bytes: You are now connected */
DO_WSK_Expect(S1, "d");

// used to store the data that was received by the DO_WSK_Expect
temp = Response();

//used to get the size of the response that was received so far.
x = ResponseLength();

//The line below will print the length of the response  to the playerbuffer
RR__printf("The size of the received buffer was %d bytes ",x,);

DO_WSK_Closesocket(S1);
```

Conclusion

In this example, Response() and ResponseLength() are used to print the message "The size of the received buffer was 21 bytes" to the player buffer window.

Accessing Server Replies with DO_WSK_Read

When the data received is too dynamic and there is nothing on which to base the unique string,you cannot use the DO_WSK_Expect() to access server replies. In this case, when the number of characters received is always the same, you can use DO_WSK_Read().

In the following example, the user sends a logon string to the server and the server sends back a unique key that is used for subsequent calls. Since the ending characters always change, DO_WSK_Read is called to receive a specified number of characters. You can parse this data for any values you need. The required code is shown in bold.

Sample Script

```
...
BEGIN_TRANSACTION();

DO_WSK_Socket(S2, AF_INET, SOCK_STREAM, IPPROTO_TCP);

DO_WSK_Bind(S2, ANY_ADDR, ANY_PORT);

DO_WSK_Connect(S2, "10.4.26.24", 80, AF_INET);

/* 20 bytes */
DO_WSK_Send(S2, "^@user=Bob^@pwd=CPWR^@\377");

//The data that is being sent from the server is dynamic and the ending
// character is never the same. Since this is the case using
// DO_WSK_Expect() will not work, so the script will use DO_WSK_Read()
// to receive the same number of bytes that were originally sent.
/* 14 bytes: SID=1234567890 */
//DO_WSK_Expect(S2, "^@");

//The script will now print the response that was received by
// calling Response() to gain access to the received buffer.

If ( DO_WSK_Read(S2,14) != -1)
RR__printf("String = %s",Response());
...
```

Conclusion

Using DO_WSK_Read() instead of DO_WSK_Expect() allows the virtual user to receive a specific number of bytes instead of a sequence of characters.

Accessing Server Replies with DO_WSK_Recv

When the data returned is too dynamic and the DO_WSK_Expect() fails, you can use DO_WSK_Recv() to store the reply returned from the server. This saves the response based on its size instead of on the unique character string used in the DO_WSK_Expect(). When you use DO_WSK_Recv(), you specify how much data you want to receive and where to store it. In this sample, the required code is shown in bold.

Sample Script

```
/* Declare Variables */
//Below are variables that were declared to use with DO_WSK_Recv,
//and the size of temp will vary depending upon the buffers that your
//application is returning.
int size = 0;
char temp[45];
…
BEGIN_TRANSACTION();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);

DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//Below is the data that was received at capture time.
/* 21 bytes: You are now connected */

//Initializing the temp variable
memset(temp,'\0',45);

//Instead of calling DO_WSK_Expect() to receive the data, the
//script will now call DO_WSK_Recv(). This will allow the script to
//receive a specified number of characters instead of looking at the data
//to determine if the buffer has been received entirely or not.

DO_WSK_Recv(S1,temp,45,0,&size);

//Original function that was in the script
//DO_WSK_Expect(S1, “d”);

//Prints the size of the string that was received to the playerbuffer.
RR__printf("Size of the received buffer was %d bytes", size);

DO_WSK_Closesocket(S1);
```

Conclusion

In this example, the message “Size of the received buffer was 21 bytes” is printed to the player buffer window.

> Caution: When using this method instead of the DO_WSK_Expect(), verify that you receive the correct information before moving on to the next function in your script.

Parsing Server Replies Using ScanFloat and ScanInt

Sometimes local applications must interpret and act on numeric values sent from a remote application. For example, remote applications may send a port number to the local application. The local application parses the remote port from the data it receives and attempts to connect to the remote machine on the new port.

By using ScanInt(), you can parse the received buffer for numeric values, such as port numbers, or use ScanFloat() to parse for larger numeric values. The sample script illustrates how to use ScanFloat() and ScanInt(). The required code is shown in bold.

Sample Script

```
...
/* Declare Variables */
//The variable "port" was declared to store the port# that the remote
// application sends. The variable "nID" is used to store the unique
// ID that is sent by the remote application.
unsigned short port;
float nID;
...
BEGIN_TRANSACTION();
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_TCP);
DO_WSK_Socket(S2, AF_INET, SOCK_STREAM, IPPROTO_TCP);
DO_WSK_Connect(S1, "10.15.21.225", 33, AF_INET);

/* 11 bytes: "port=\nA\0\0aB" */

//Since the end of the data that is being received is dynamic, the
// script will use DO_WSK_Read() to receive the data instead
// using the DO_WSK_Expect() that was converted into the script.
//DO_WSK_Expect("B");
DO_WSK_Read(S1,11);
//Calling ScanSkip to move the pointer to the first byte of the port
ScanSkip(5);

//Calling ScanInt to copy the next two bytes into a variable that will
// store the port from the remote application.
ScanInt(MyByteOrder(),2,(char*)&port);

//Calling ScanFloat to copy the next four bytes into a variable that will
// store the unique ID from the remote application.
ScanFloat(MyByteOrder(),4,(char*)&nID);
//Printing the port number to the playerbuffer window, and converting
// the value from network byte order to host byte order.
```

```
RR__printf("port=%d",ntohs(port));
//Printing the unique ID that the remote application sent to the
// playerbuffer.
RR__printf("float=%f",nID);
DO_WSK_Closesocket(S1);
//Below is the original DO_WSK_Connect that was converted into the script
//DO_WSK_Connect(S2,″10.15.21.225″, 2526, AF_INET);
//Uses the port that was returned above to reconnect to the remote application
DO_WSK_Connect(S2, "10.15.21.225", ntohs(port), AF_INET);
DO_WSK_Send(S2,"Hello World");
DO_WSK_Closesocket(S2);
...
```

Conclusion

In this example, the script parses the port number sent from the remote application and uses this to reconnect to the remote application. The script also uses ScanFloat() to parse a unique ID from the message sent by the remote application.

Parsing Server Replies Using ScanSkip and ScanString

Data returned from the server may be too dynamic to base your parsing on actual characters. In this case, you can base your search on character positions using ScanSkip() and ScanString().

For example, you can save characters 20 through 25 that are returned from a server. The ScanSkip() skips the specified number of characters in the internal buffer that stores the response received in the DO_WSK_Expect(). The ScanString() scans the number of characters you specify into a character string from the current position in the buffer.

In this sample, the buffer returned from the server is xxx123456789yyy. You are retrieving the value between xxx and yyy. The required code is shown in bold.

Sample Script

```
...
/* Declare Variables */
//Variable to store the string that we are searching for.
char temp[15];
…
BEGIN_TRANSACTION();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
```

```
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//Below is the actual data that was returned at capture time.
/* 16 bytes: xxx0123456789yyy */
DO_WSK_Expect(S1, "yyy");
//making sure that all of the data within our variable has been initialized
memset(temp,'\0',15);
//skips 3 bytes within the buffer that was received.
ScanSkip(3);

//copies the next 10 bytes of the buffer that was received into our temp variable
ScanString(10,temp);
//Displays the string that was found to the playerbuffer window.
RR__printf("string=%s",temp);
DO_WSK_Closesocket(S1);
...
```

Conclusion

In this example, the message “string=0123456789” is printed to the player buffer window.

Receiving Winsock UDP Data with DO_WSK_Recvfrom

When the application you are capturing is using the UDP protocol, you can use the DO_WSK_Recvfrom() to receive data from the remote application. The sample script illustrates how to use the Do_Wsk_Recvfrom (). The required code is shown in bold.

Sample Script

```
...
/* Declare Variables */
//The variable strBuf is used to formulate the dynamic data that is
// sent in the DO_WSK_Sendto() below.
char strBuf[256];
//RemoteAddr stores the remote applications address and port
struct sockaddr_in RemoteAddr;
//nBytes is used to stor the number of bytes that were received via DO_WSK_Recvfrom()
int nBytes = 0;
…
BEGIN_TRANSACTION();
DO_WSK_Socket(S1, AF_INET, SOCK_DGRAM, IPPROTO_UDP);

DO_WSK_Bind(S1, "10.15.16.26", 333);
```

```
//Receives data via the UDP protocol
DO_WSK_Recvfrom(S1,strBuf,(struct sockaddr*) &RemoteAddr, 256, 0, &nBytes);

//Prints to the player buffer the number of bytes that were received and
// the remote address and port that sent them.
RR__printf("Received %d bytes from %s:%d",
   nBytes, inet_ntoa(RemoteAddr.sin_addr),
   ntohs(RemoteAddr.sin_port));
…
DO_WSK_Closesocket(S1);
```

Conclusion

This example shows how to use DO_WSK_Recvfrom() to receive UDP protocol data. It shows how to access the address and port of the remote application sending the data.

Sending Variable Data with DO_WSK_Send

Captured data may not be the data you want to use while running a test. For example, you might change the user name sent during capture time to a different value during replay. You can change the value in the DO_WSK_Send() to make a static value within the function. However, if you want to substitute a different value each time, you can create a dynamic variable, such as a datapool value, to replace the user name.

In this example, the script includes a DO_WSK_Send() that sends "name=Jim" to the server as the user name. For testing purposes, you want to change the name to include a variable that represents a different name, such as "Mark". The code required for this is shown in bold.

Sample Script

```
…
/* Declare Variables */
//Below are variables needed for the dynamic data. The size of these
//variables will depend upon how big the buffer is that you are replacing
char buffer[65];
char sendbuffer[65];
…
BEGIN_TRANSACTION();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);

DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

//Below is the original send based upon the capture file.
```

```
// DO_WSK_Send(S1,"name=Jim");
//The script will now create a buffer that is different than what was captured,
//and if we wanted the string "Mark" could also of been a datapool value.
strcpy( buffer, "Mark");
sprintf( sendbuffer, "name=%s", buffer);
//The script will now send the variable string that was created above
DO_WSK_Send(S1, sendbuffer);
/* 2 bytes: ok */
DO_WSK_Expect(S1,"ok");
DO_WSK_Closesocket(S1);
```

Conclusion

In this example, the user name sent to the server is changed from "name=Jim" to "name=Mark" by modifying the buffer before the DO_WSK_Send() and passing the new buffer as the second parameter of the DO_WSK_Send().

Sending Variable Data using DO_WSK_SendAll

When only a portion of a string that the script is sending must be modified to make it dynamic, using the DO_WSK_SendAll() can be easier than modifying the DO_WSK_Send().

The following example sends the username "Bob" and the password "CPWR" to the server to logon. Since only one instance of this user can be logged on to the server, you must modify the script to read different user names and passwords from a datapool before sending it. In the sample script snippet, the required code is shown in bold.

Sample Script

```
...
BEGIN_TRANSACTION();
//Reads in a datapool record to be used to make the username and password dynamic.
GET_DATA();
DO_WSK_Socket(S2, AF_INET, SOCK_STREAM, IPPROTO_TCP);

DO_WSK_Bind(S2, ANY_ADDR, ANY_PORT);

DO_WSK_Connect(S2, "10.0.6.32", 80, AF_INET);

//In the string that was sent below "Bob" and "CPWR" both need to be modified
// to allow for multiple users to execute this script.
/* 20 bytes */
//DO_WSK_Send(S2, "^@user=Bob^@pwd=CPWR^@\377");

//Using DO_WSK_Sendall the message that is being sent can easily be modified, so
// that the script can read in the username and password from a datapool file.
```

```
DO_WSK_SendAll(S2,5, "^@user", VARDATA(1), "^@pwd=", VARDATA(2), "^@\377");
/* 15 bytes: SID=1234567890^@ */
DO_WSK_Expect(S2, "^@");

DO_WSK_Closesocket(S2);
...
```

Conclusion

In the sample modified script, the user name and password are read in from a datapool file and the data is sent using the DO_WSK_SendAll().

Sending Variable Winsock UDP Data

Captured data may not be the data that you want to use while running a test. For example, you may want to substitute a user name sent during capture with a different value each time during replay. You can do this using a dynamic variable, such as a datapool variable.

Note: Changing the value located in the DO_WSK_Sendto() makes the value static within the function.

In this sample, the code in bold below shows how change data values.

Sample Script

```
...
/* Declare Variables */
//The variable strBuf is used to formulate the dynamic data that is
// sent in the DO_WSK_Sendto() below.
char strBuf[24];
…
BEGIN_TRANSACTION();
//Reads in a row of data from a central datapool file, and this will be
//used as part of the dynamic data that will be sent via the DO_WSK_Sendto()
GET_DATA();
DO_WSK_Socket(S1, AF_INET, SOCK_DGRAM, IPPROTO_IP);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_BROADCAST, 1);

DO_WSK_Bind(S1, "127.0.0.1", 5634);

//Below is the original DO_WSK_Sendto() that was converted into the script
//DO_WSK_Sendto(S1, "name=Brian", "10.25.26.24", 1234);
//The example below reads a record in from a central datapool, and the value
// is then sent as part of the DO_WSK_Sendto() buffer.
memset(strBuf, 0, 24);
sprintf(strBuf, "name=%s", VARDATA(1));
DO_WSK_Sendto(S1, strBuf, "10.25.26.24",  1234);
```

```
…
DO_WSK_Closesocket(S1);
…
```

Conclusion

In this example, the virtual user reads in a row of data from a central datapool, which is used to create a dynamic message. This message is used in the DO_WSK_Sendto() to send it to the address "10.25.26.24".

Sending Variable Data using DO_WSK_Write

You can use DO_WSK_Write() instead of DO_WSK_Send() when coding scripts by hand. DO_WSK_Write() does not expect strings that have certain control and null characters encoded, as does DO_WSK_Send(). This allows you to send data without using EscapeStr() to encode any possible control characters.

The sample script illustrates how to use DO_WSK_Write(). The required code is shown in bold.

Sample Script

```
...
/* Declare Variables */
//Variable that will be used to send data to the server via DO_WSK_Write()
char temp[24];
...
BEGIN_TRANSACTION();
DO_WSK_Socket(S2, AF_INET, SOCK_STREAM, IPPROTO_TCP);
DO_WSK_Bind(S2, ANY_ADDR, ANY_PORT);
DO_WSK_Connect(S2, "10.4.26.24", 60, AF_INET);

//The original data that was sent was... "^@user=Bob^@pwd=CPWR^@", and the
// ^@ are encoded null characters. The function below constructs the same
// data that was sent without the data being encoded.
memcpy(temp,"\0user=Bob\0pwd=CPWR\0", 19);
/* 19 bytes */
//The DO_WSK_Send() below was converted to the script from the capture file.
//DO_WSK_Send(S2, "^@user=Bob^@pwd=CPWR^@");

//Since the script is now using DO_WSK_Write() to send the data it does not
//need to call EscapeStr() to encoded the NULL characters within the string
//that is being sent.
DO_WSK_Write(S2,temp, 19);
/* 15 bytes: SID=1234567890^@ */
DO_WSK_Expect(S2, "^@");
DO_WSK_Closesocket(S2);
...
```

Conclusion

In the sample script, the data was sent using the DO_WSK_Write() instead of the DO_WSK_Send(). This allows you to send data without encoding it.

Using Central Datapools within a Winsock Script

You can use dynamic data in your script by reading data from a datapool file. However, datapool files must be in an ASCII string, and not all dynamic data are in this format. For example, when the string "\121\101\114\157\141\144" appears in a datapool file and is read in using a one of the datapool functions, you receive \\121\\101\\114\\157\\141\\144 as the output.

You can ensure that the output string you receive is accurate by using the OctalToChar() to convert any octal sequences into their binary representation. In this sample script, the string "\121\101\114\157\141\144" is read in from a central datapool file and converted to its binary representation. The required code is shown in bold.

Sample Script

```
/* Declare Variables */
//Variable declared to store the central datapool record.
char temp[40];
…
BEGIN_TRANSACTION();
//Gets a row of data from the Conductor during the test.
GET_DATA();
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
//Read the first column from the central datapool row that was
//returned when the GET_DATA() was called.
strcpy(temp,VARDATA(1));

//used to convert octal strings to their binary format
OctalToChar(temp);
//The script is now setup to send the data that was read in from the
//central datapool file instead of the hard coded values that were captured.
DO_WSK_Send(S1,temp);
//Below is the original send that was captured.
//DO_WSK_Send(S1,″\121\101\122\165\156″);
DO_WSK_Closesocket(S1);
```

Conclusion

In this example, the DO_WSK_Send() sends the octal representation for the string "QALoad", "\121\101\114\157\141\144", to the server.

Using Local Datapools within a Winsock Script

You can use dynamic data in your script by reading data from a datapool file. However, datapool files must be in an ASCII string, and not all dynamic data are in this format. For example, when the string "\121\101\114\157\141\144" appears in a datapool file and is read in using a one of the datapool functions, you receive \\121\\101\\114\\157\\141\\144 as the output.

You can ensure that the output string you receive is accurate by using the OctalToChar() to convert any octal sequences into their binary representation. In this sample script, the string "\121\101\114\157\141\144" is read in from a local datapool file and converted to its binary representation. The required code is shown in bold.

Sample Script

```
//This is used as an easy to remember descriptor for the datapool file.
#define DP1 1/* Identifier for a datapool file*/
/* Declare Variables */
//Variable declared to store the local datapool record.
char temp[40];
…
//Opens the datapool file
OPEN_DATA_POOL("datapool.dat", DP1, TRUE)
BEGIN_TRANSACTION();
//Reads a row of data in from the datapool file that was opened above
READ_DATA_RECORD(DP1);
…
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);

DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);

DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
//Read the first column from the local datapool row that was
//returned when the READ_DATA_RECORD () was called.
strcpy(temp, GET_DATA_FIELD(DP1, 1));

//used to convert octal strings to their binary format
OctalToChar(temp);
//The script is now setup to send the data that was read in from the
//local datapool file instead of the hard coded values that were captured.
DO_WSK_Send(S1,temp);
//Below is the original send that was captured.
//DO_WSK_Send(S1,″\121\101\122\165\156″);
```

```
DO_WSK_Closesocket(S1);
```

Conclusion

In this example, the DO_WSK_Send() sends the octal representation for the string “QALoad”, ““\121\101\114\157\141\144”, to the server.

WWW Scripts

WWW Script Samples

You can address specific situations or resolve certain problems by modifying converted WWW scripts. The samples shown here include a description of the problem, the procedure for implementing the modification, and samples of a modified script. Scripts modifications are discussed for:

Extracting a String from a WWW Response and Reusing it as a CGI Parameter

Extracting and Reusing Web Service XML Values

Moving the Transaction Loop

Extracting and Reusing Cookies

Extracting a String from a WWW Response for Validation

Forcing a Subrequest

IP Spoofing with a Local Datapool

Preventing Unwanted Subrequests

Extracting a String from a WWW Response for Validation

Overivew: Extracting a String from WWW Response for Validation

In some load tests, you must validate an operation by examining a string other than the document title. In these cases, you can extract the string and use a comparison function.

Scripting

When you script for string extraction and validation, you must:

- Create the variable to store the extracted string value.
- Modify the script to extract the string from the response page.
- Compare the extracted string to the expected result.

Visual Navigator

If you are using Visual Navigator, you can use the Content Check in the tree view to specify any text that should be present in the response.

Conclusion

Using these scripting techniques, you can modify a script for string extraction and validation. The sample script segment illustrates the use of extraction and comparison.

Sample: Extracting a String from WWW Response for Validation

The following sample is a script segment that illustrates the use of extraction and comparison.

Sample Script

```
.
.
.
// Declare variables
CloadString strName = "";
Navigate_To("http://test.page.com");
Click_on(LINK, "First Test");
/*
 *
 * This page should return HTML in this form
 ********************************************
 <HTML>
 <HEAD>Test Page</HEAD>
 <BODY>
 Welcome to the first day of testing, James.<br>
 You have 42 tasks left to complete.
 </BODY>
 </HTML>
 ********************************************
 *
* Passing the test requires that the name on
* the page is James and not Jim.
 *
 */
strName = Get(REPLY, STRING, "testing, ", ".<br>");
if ( strName.Compare("James") != 0)
{
RR__FailedMsg(s_info, "James was not returned");
}
.
.
.
```

Extracting a String from a WWW Response to Reuse as a CGI Parameter

Overview: Extracting and Reusing a String as a CGI Parameter

Page requests sometimes use CGI parameter values received in a previous page response. Since the value is received dynamically, the script must extract that value from the previous response and insert it into the CGI parameter list at runtime.

Scripting

When you script for string extraction and reuse, you must:

- Create the variables to store the extracted string value.
- Modify the script to extract the string values from the page response.
- Insert the variables into the CGI parameter list.

Visual Navigator

When you use Visual Navigator, you can use the Parameterization Rules Wizard for this task.

Conclusion

By following these scripting techniques, you can modify a Visual Script to extract values from page requests and reuse the string's CGI parameter values in later page requests. You can use similar scripting techniques for reusing extracted strings in form values.

The sample original script and the sample modified script illustrate the procedure for extracting and reusing a string.

Sample: Original WWW Script Snippets

The following are original script snippets from a record session before extracting a string to reuse as a CGI parameter. The sample below has two page requests. The first request returns HTML with values that must be extracted for use with the second page request. The second page request uses form values to pass the CGI parameters. The comments indicate the locations in the script where the scripting steps occur. Points of interest in the script are highlighted in bold

Sample Script Snippets

```
... beginning of script ...

extern "C" int rhobot_script(PLAYER_INFO* s_info)

{

   // Declare Variables

   //

   // Add variables to receive extracted strings here!

   // NOTE: Declare the variable as CLoadString – this

   //     class is used by Visual Script to encapsulate

   //     string functionality.

   SET_ABORT_FUNCTION(abort_function);

... script up to Request #2 ...

   //---------  REQUEST # 2 ---------

   //

   //  current page url is http://qawebserv/cgiquery.htm

   //

   Click_On(LINK, 1, DESCRIPTION, "CGI Query Example");

   Verify(PAGE_TITLE, "CGI Query Example");
```

```
// The response page for this request will contain
// the values we want to extract. Each value will
// be extracted using the Get command with the STRING attribute
// set and the left (preceded-by string) and
// right (followed-by string) as parameters.
/*** HTML snippet with values we're looking to extract ***
     <body><h2>Get Query Value Result</h2>
     Name: JohnDoe <br>
     Password: jdoe<br>
     E-mail Address: johndoe@compuware.com<br>
     Address: One Main Street<br>
     City: Anywhere<br>
     State: NY<br>
     Zip Code: 12345<br>
     Comments:This is an example of string extraction in WWW<br>
     <hr>
*** /
//---------  REQUEST # 3  (see action item on Page 6)  ---------
//
//  current page url is http://qawebserv/cgiquery.htm
//
// The values set for the CGI parameter in the Set command
// will need to be replaced with variables holding the
// extract string values.
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "name", "JohnDoe");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "password", "jdoe");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "e-mail",
"johndoe@compuware.com");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "address",
"One Main Street");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "city", "Anywhere");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "state", "NY");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "zip", "12345");
Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "comments",
"This is an example of string extraction in WWW");

Click_On(BUTTON, 1, DESCRIPTION, "Submit CGI Query");
Verify(PAGE_TITLE, "CGI Query - Results");
... end of script...
```

Sample: Modified WWW Script Snippets

The following sample is a modified WWW script snippet that extracts a string to reuses it as a CGI parameter. Changes to original script are highlighted in bold.

Sample Script Snippet

```
... beginning of script...

extern "C" int rhobot_script(PLAYER_INFO* s_info)

{

   // Declare Variables

   //

   CLoadString strName = "";

   CLoadString strEmailAddress = "";

   CLoadString strPassword = "";

   CLoadString strAddress = "";

   CLoadString strCity = "";

   CLoadString strState = "";

   CLoadString strZip = "";

   CLoadString strComment = "";

   SET_ABORT_FUNCTION(abort_function);

... script up to Request #1...

   //---------  REQUEST # 2 ---------

   //

   //  current page url is http://qawebserv/cgiquery.htm

   //

   Click_On(LINK, 1, DESCRIPTION, "CGI Query Example");

   Verify(PAGE_TITLE, "CGI Query Example");


   strName = Get(REPLY, STRING, "Name: ", " <br>");

   strPassword = Get(REPLY, STRING, "Password: ", "<br>");

   strEmailAddress = Get(REPLY, STRING, "E-mail Address: ", "<br>");

   strAddress = Get(REPLY, STRING, "<br>\rAddress: ", "<br>");

   strCity = Get(REPLY, STRING, "City: ", "<br>");

   strState = Get(REPLY, STRING, "State: ", "<br>");

   strZip = Get(REPLY, STRING, "Zip Code: ", "<br>");

   strComment = Get(REPLY, STRING, "Comments: ", "<br>");

   //---------  REQUEST # 3  (see action item on Page 2)  ---------

   //

   //  current page url is http://qawebserv/cgiquery.htm

   //

   Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "name", __Name);

   Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "password", __Password);
```

```
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "e-mail", __EmailAddress);
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "address", __Address);
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "city", __City);
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "state", __State);
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "zip", __Zip);
    Set (NEXT_REQUEST_ONLY, CGI_PARAMETER, "comments", __Comment);

    Click_On(BUTTON, 1, DESCRIPTION, "Submit CGI Query");
    Verify(PAGE_TITLE, "CGI Query - Results");
... end of script...
```

Extracting and Reusing Cookies

Overview: Extracting and Reusing a Cookie

In some load tests, you may need to extract a cookie and use it later in the script.

Scripting

To extract the cookie, use the DO_GetCookie(). Reuse the value with DO_SetValue.

Conclusion

The sample script snippet illustrates extracting and reusing a cookie.

Sample: Extracting and Reusing a Cookie

The following sample is a portion of a script that extracts and reuses a cookie.

Sample Script

```
char * userid;
char * aspsessionid;
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http ("GET http://company.com/ HTTP/1.0\r\n\r\n");
/*
 * Get a cookie named USER_ID
 */
DO_GetCookie ( "USER_ID", 1, &userid );
/*
 * Get the second ASPSESSIONID cookie. ASPSESSIONID
```

```
 * cookies always have extra characters on the end to make
 * them unique.
 *
 * An example ASPSESSIONID: ASPSESSIONIDQQQGGQDO=EBOOONBBFH
 * BBELAJIMEFAKAP
 */
DO_GetCookie ("ASPSESSIONID*", 2, &aspsessionid );

DO_SetValue("User", userid)

DO_Http("POST http://company.com/ HTTP/1.0\r\n\r\n"
 "Content-Type: application/x-www-form-urlencoded\r\n"
 "Content-Length: {*content-length}\r\n"
        "{User}");
```

Extracting and Reusing Web Service XML Values

Overview: Extracting and Reusing Web Service XML Values

When testing Web services, data dependencies may exist between XML request data values and XML values returned as part of an XML document reply earlier in the script. You can retrieve values from a response containing an XML document and store the values in a variable, by using a version of the Get command. You can reuse this value as a CGI parameter, as post data, or you can substitute it into an XML document string sent as part of a Web Services XML request.

To modify the script:

1. Extract an XML value by inserting a Get command in the script after the request.
2. Ensure that the XPATH attribute is set and that the XML element is specified. Use the modified XPath syntax used by QALoad to extract text and attribute values from an XML document response page.

Visual Navigator

Visual Navigator provides a GUI interface in the Visual tree pane. You can use this to extract text and attribute values from XML documents returned as a reply, and to parameterize text and attribute values sent as part of an XML request.

Conclusion

By following these techniques, you can extract XML text and attribute values from XML documents into variables for later use.

The sample original script and the sample modified script illustrate these techniques. The modified script reuses the value in a subsequent XML request.

Sample: Original WWW Script for Web Services

The following sample shows original script snippets converted from a record session. Points of interest in the script are highlighted in bold. Bold and italicized text indicates script lines that will be modified.

Sample Script Snippets

```
... beginning of script ...
```

```
extern "C" int rhobot_script(PLAYER_INFO* s_info)
{
   // Declare Variables
   //
   CLoadString _QALTmpVars[  4];
   // Create and initialize the CLoadString variable to
   // hold the XML value.
... more script to Request 1...
   //---------  REQUEST # 1  ---------
   //
   Set (NEXT_REQUEST_ONLY, HEADER, "SOAPAction",
   "\"http://tempuri.org/Calc/action/Calc.Add\"");
   _QALTmpVars[0]  = "<?xml version=\"1.0\" encoding=\"UTF-8\"
      standalone=\"no\"?>";
   _QALTmpVars[0] += "<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=\"\"
      xmlns:SOAPSDK1="
      "\"http://www.w3.org/2001/XMLSchema\" xmlns:SOAPSDK2="
      "\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:SOAPSDK3="
      "\"http://schemas.xmlsoap.org/soap/encoding/\"
      xmlns:SOAP-ENV="
           "\"http://schemas.xml";
   _QALTmpVars[0] += "lsoap.org/soap/envelope/\">";
   _QALTmpVars[0] += "<SOAP-ENV:Body SOAP-ENV:encodingStyle="
        "\"http://schemas.xmlsoap.org/soap/encoding/\">";
   _QALTmpVars[0] +=  "<m:Add
   xmlns:m=\"http://tempuri.org/Calc/message/\" "
         "SOAP-ENV:encodingStyle=\"\">";
   _QALTmpVars[0] +=   "<A SOAP-ENV:encodingStyle=\"\">10</A>";
   _QALTmpVars[0] +=   "<B SOAP-ENV:encodingStyle=\"\">5</B>";
   _QALTmpVars[0] +=  "</m:Add>";
   _QALTmpVars[0] += "</SOAP-ENV:Body>";
   _QALTmpVars[0] += "</SOAP-ENV:Envelope>";

   Set(NEXT_REQUEST_ONLY, XML_DATA, (char *)_QALTmpVars[0]);
   XmlRequest( "POST",
          "http://mssoapsampleserver/MSSoapSamples30/Calc/Service/SrSz/AspVbsCpp/Calc."
           "asp" );
   // An XML text value from the XML document returned from
   // Request 1 will be extracted here using an inserted Get command.
```

```
//---------  REQUEST # 2  ---------
//
Set (NEXT_REQUEST_ONLY, HEADER, "SOAPAction",
 "\"http://tempuri.org/Calc/action/Calc.Multiply\"");
// Visual Scripting takes XML documents sent as part of a
// XML or Web Service request and stores it in a CLoadString
// variable.  Inserting a value means altering the string
// concatenation for the literal value and
// substituting the variable.
_QALTmpVars[1]  = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?>";
_QALTmpVars[1] += "<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=\"\"
xmlns:SOAPSDK1="
      "\"http://www.w3.org/2001/XMLSchema\" xmlns:SOAPSDK2="
      "\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:SOAPSDK3="
          "\"http://schemas.xmlsoap.org/soap/encoding/\" xmlns:SOAP-ENV="
"\"http://schemas.xm";
_QALTmpVars[1] += "lsoap.org/soap/envelope/\">";
_QALTmpVars[1] += "<SOAP-ENV:Body SOAP-ENV:encodingStyle="
     "\"http://schemas.xmlsoap.org/soap/encoding/\">";
_QALTmpVars[1] +=  "<m:Multiply
     xmlns:m=\"http://tempuri.org/Calc/message/\" "
     "SOAP-ENV:encodingStyle=\"\">";
// We want to substitute the result from Request 1
// for the text value "10" for the "A" tag.
_QALTmpVars[1] +=   "<A SOAP-ENV:encodingStyle=\"\">10</A>";
_QALTmpVars[1] +=   "<B SOAP-ENV:encodingStyle=\"\">5</B>";
_QALTmpVars[1] +=  "</m:Multiply>";
_QALTmpVars[1] += "</SOAP-ENV:Body>";
_QALTmpVars[1] +="</SOAP-ENV:Envelope>";
Set(NEXT_REQUEST_ONLY, XML_DATA, (char *)_QALTmpVars[1]);
XmlRequest( "POST",
"http://mssoapsampleserver/MSSoapSamples30/Calc/Service/SrSz/AspVbsCpp/Calc.asp" );
... rest of script ...
```

Sample: Modified WWW Script for Extracting and Reusing Web Service XML Values

The following sample is a script snippet that is modified to extract and reuse Web service XML values. Code added to the script is highlighted in bold. Modified code is in bold and italicized.

Sample Script Snippet

```
... beginning of script ...
extern "C" int rhobot_script(PLAYER_INFO* s_info)
{
   // Declare Variables
   //
   CLoadString _QALTmpVars[  4];
   CLoadString __Sum = "";
... more script to Request 1...
   //---------  REQUEST # 1  ---------
   //
   Set (NEXT_REQUEST_ONLY, HEADER, "SOAPAction",
   "\"http://tempuri.org/Calc/action/Calc.Add\"");
   _QALTmpVars[0]  = "<?xml version=\"1.0\" encoding=\"UTF-8\"
   standalone=\"no\"?>";
   _QALTmpVars[0] += "<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=\"\"
   xmlns:SOAPSDK1="
   "\"http://www.w3.org/2001/XMLSchema\" xmlns:SOAPSDK2="
   "\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:SOAPSDK3="
   "\"http://schemas.xmlsoap.org/soap/encoding/\"
   xmlns:SOAP-ENV="
        "\"http://schemas.xml";
   _QALTmpVars[0] += "lsoap.org/soap/envelope/\">";
   _QALTmpVars[0] += "<SOAP-ENV:Body SOAP-ENV:encodingStyle="
        "\"http://schemas.xmlsoap.org/soap/encoding/\">";
   _QALTmpVars[0] +=  "<m:Add
   xmlns:m=\"http://tempuri.org/Calc/message/\" "
         "SOAP-ENV:encodingStyle=\"\">";
   _QALTmpVars[0] +=   "<A SOAP-ENV:encodingStyle=\"\">10</A>";
   _QALTmpVars[0] +=   "<B SOAP-ENV:encodingStyle=\"\">5</B>";
   _QALTmpVars[0] +=  "</m:Add>";
   _QALTmpVars[0] += "</SOAP-ENV:Body>";
   _QALTmpVars[0] += "</SOAP-ENV:Envelope>";

   Set(NEXT_REQUEST_ONLY, XML_DATA, (char *)_QALTmpVars[0]);
   XmlRequest( "POST",
          "http://mssoapsampleserver/MSSoapSamples30/Calc/Service/SrSz/AspVbsCpp/Calc."
           "asp" );
   strSum = Get ( XML, TEXT,
"/SOAP-ENV:Envelope[1]/SOAP-ENV:Body[1]/m:AddResponse[1]/Result[1]" );
```

```
//---------  REQUEST # 2  ---------
//
Set (NEXT_REQUEST_ONLY, HEADER, "SOAPAction",
 "\"http://tempuri.org/Calc/action/Calc.Multiply\"");
_QALTmpVars[1]  = "<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?>";
_QALTmpVars[1] += "<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=\"\"
xmlns:SOAPSDK1="
      "\"http://www.w3.org/2001/XMLSchema\" xmlns:SOAPSDK2="
      "\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:SOAPSDK3="
          "\"http://schemas.xmlsoap.org/soap/encoding/\" xmlns:SOAP-ENV="
"\"http://schemas.xm";
_QALTmpVars[1] += "lsoap.org/soap/envelope/\">";
_QALTmpVars[1] += "<SOAP-ENV:Body SOAP-ENV:encodingStyle="
"\"http://schemas.xmlsoap.org/soap/encoding/\">";
_QALTmpVars[1] +=  "<m:Multiply
xmlns:m=\"http://tempuri.org/Calc/message/\" "
"SOAP-ENV:encodingStyle=\"\">";
// Original value: <A SOAP-ENV:encodingStyle=\"\">10</A>
// Variablized value: <A SOAP-ENV:encodingStyle=\"\">{$ VAR:Sum $}</A>
_QALTmpVars[1] +=   "<A SOAP-ENV:encodingStyle=\"\">";
_QALTmpVars[1] +=                                __Sum;
_QALTmpVars[1] +=   "</A>";
_QALTmpVars[1] +=   "<B SOAP-ENV:encodingStyle=\"\">5</B>";
_QALTmpVars[1] +=  "</m:Multiply>";
_QALTmpVars[1] += "</SOAP-ENV:Body>";
_QALTmpVars[1] +="</SOAP-ENV:Envelope>";
Set(NEXT_REQUEST_ONLY, XML_DATA, (char *)_QALTmpVars[1]);
XmlRequest( "POST",
"http://mssoapsampleserver/MSSoapSamples30/Calc/Service/SrSz/AspVbsCpp/Calc.asp" );
... rest of script ...
```

Forcing a Subrequest

Overview: Forcing a Subrequest that is not Being Made Automatically

Occasionally, one of the necessary subrequests that is recorded is not requested automatically at playback time. This can be caused by complex javascript code not executing correctly, or by an ActiveX control that cannot be used at playback time. You can force this subrequest by inserting an ADDITIONAL_SUBREQUEST statement.

Scripting

To request an additional subrequest, insert a command like the one below before the action statement (Click_On, Navigate_To, or Post_To).

```
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST, "http://xyz.com/onsale.aspx");
```

This adds the specified URL to the list of subrequests automatically generated when the requested page is parsed.

Conclusion

Using these techniques, you can modify a script to force a subrequest. The sample modified script illustrates forcing a subrequest.

Sample: Modified Script for Forcing a Subrequest

In this example, one of the pages has an ActiveX object associated with it that automatically generates a request for items that are on sale. However, at playback this request is not made because ActiveX objects are not executed at playback time. To force this subrequest, you can insert an Additional Subrequest item just before the action statement. Relevant statements are shown in bold.

Sample Script

```
//---------  REQUEST # 5  (see action item on Page 4)  ---------
//
//  current page url is http://xyz.com/chairs.htm
//
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST, "http://xyz.com/onsale.aspx");
Click_On(LINK, 1, DESCRIPTION, "Nuts and Bolts");
Verify(PAGE_TITLE, "Nuts and Bolts");
```

IP Spoofing with a Local Datapool

Overview: IP Spoofing with a Local Datapool

Datapools can provide IP spoofing addresses to scripts on playback machines. By creating a special IP spoofing datapool for the particular playback machine, you can spoof the correct addresses at runtime.

> Note: For the script to execute properly, the IP addresses in the local datapool must match the IP addresses bound to the network cards on the playback machine.

Scripting

When you modify a Visual Script to allow for IP spoofing for a local playback machine, you must:

- Create a datapool file with the IP addresses associated with the network cards on the playback machine.
- Insert the required scripting code for datapool access into the script.
- Retrieve and use the spoofed IP address.

Visual Navigator

Visual Navigator provides GUI interfaces in the tree view for creating datapool files and using spoofed IP addresses. You can use the datapool variables as IP spoofing addresses in the IP spoof interface.

Conclusion

By following these scripting techniques, you can modify a Visual Script to extract IP addresses from datapool files for use as spoofed addresses. The modified script sample illustrates the modifications required in the script.

Sample: Modified Script for IP Spoofing with a Local Datapool

This sample snippet shows the scripting required to open and access the datapool and use the extracted IP address as a spoofed address. Points of interest and modified code in the script are highlighted in bold.

Sample Script Snippet

```
... beginning of script ...

   // Before the SYNCHRONIZE command, add a #define to
   // identify the datapool file and a call to
   // OPEN_DATA_POOL to open the datapool file
   // (Note: the datapool file name is "ipspoof.dat")
   #define IPSPOOF_DP  1  /* identifier for datapool file */
   OPEN_DATA_POOL("ipspoof.dat", IPSPOOF_DP, TRUE);
   SYNCHRONIZE();
   BEGIN_TRANSACTION();
      RESTART_TRANSACTION_TOP(); // do not modify this statement
   // After the start of transaction, read the data record, and
   // use the Set command to specify that this value is to
   // be used as the spoofed IP address for every request.
      READ_DATA_RECORD(IPSPOOF_DP);
      Set (EVERY_REQUEST, SPOOFED_IP_ADDRESS,
   GET_DATA_FIELD (IPSPOOF_DP, 1) );
... rest of script ...
```

Moving the Transaction Loop Statements

Overview: Moving the Transaction Loop Statements

Load testing a web site can require you to logon and logout when you want to perform multiple transactions during a single session. You can move the BEGIN_TRANSACTION statement and the END_TRANSACTION statement so that the virtual user does not logon and logout with every transaction.

To modify the script:

1. Move the BEGIN_TRANSACTION and RESTART_TRANSACTION_TOP statements so that they are just below any statements that log the user onto the system.
2. Keep the RESTART_TRANSACTION_BOTTOM, Clear(TRANSACTION), and END_TRANSACTION statements together and move them so that they are immediately above the Logout requests.

Visual Navigator

If you are using Visual Navigator, you can move the Transaction Loop statements in the tree view rather than in the C++ script. Simply select the Begin Transaction item and click the Move Down button until it moves below the Logon pages. Do the same for the End Transaction by selecting the Transaction Cleanup tree item and moving it. Any pages after the Transaction Cleanup, such as the Logout, take place after all transaction in the main loop have finished executing.

Sample Scripts

The following original script sample and modified script sample show a script that logs into a newsgroup forum, performs actions, such as reading threads and replying to them, and then logs out. The scripts identify the Logon and Logout requests and show the old and new locations of the transaction statements.

Sample: Original WWW Script Showing the Transaction Loop

The following sample shows the original script. The sections in italics are the requests involved with the Login and Logout. The original locations of the transaction statements are shown in bold.

Sample Script

```
SYNCHRONIZE();

BEGIN_TRANSACTION();

RESTART_TRANSACTION_TOP(); // do not modify this statement


//---------  REQUEST # 1  ---------

//

Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 1 - ");


Navigate_To("http://www.myforum.org/");

Verify(PAGE_TITLE, "My woodworking forum - powered by vBulletin");


DO_SLEEP(9);


//---------  REQUEST # 2  (see action item on Page 1)  ---------

//

//  current page url is http://www.myforum.org/

//

// The submit button for this form could not be positively identified so

// an appropriate action has been added (Post_To or Navigate_To).  The

// values for the form fields will be set using Set( ) statement.

//

Set(NEXT_REQUEST_ONLY, POST_DATA, "vb_login_username", "John Smith");

Set(NEXT_REQUEST_ONLY, POST_DATA, "cookieuser", "1");

Set(NEXT_REQUEST_ONLY, POST_DATA, "vb_login_password", "");

Set (NEXT_REQUEST_ONLY, POST_DATA, "s", "3669625a913735e99e31ec9a866a2f28");

Set (NEXT_REQUEST_ONLY, POST_DATA, "do", "login");

Set (NEXT_REQUEST_ONLY, POST_DATA, "forceredirect", "1");
```

```
Set (NEXT_REQUEST_ONLY, APPEND_CRLF_AFTER_POST_BODY, TRUE);
Set (NEXT_REQUEST_ONLY, POST_DATA, "vb_login_md5password",
       "df0349ce110b69f03b4def8012ae4970");
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 2 - ");

Post_To("http://www.myforum.org/login.php");
Verify(PAGE_TITLE, "My Woodworking Forum");

DO_SLEEP(3);

//---------  REQUEST # 4  (see action item on Page 3)  ---------
//
//  current page url is http://www.myforum.org/
//
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 4 - ");

Click_On(LINK, 1, DESCRIPTION, "Tables and Chairs");
Verify(PAGE_TITLE, "My Woodworking Forum – Tables and Chairs");

DO_SLEEP(2);

//---------  REQUEST # 5  (see action item on Page 4)  ---------
//
//  current page url is http://www.myforum.org/forumdisplay.php?f=17
//
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 5 - ");

Click_On(LINK, 1, DESCRIPTION, "Log Out");
Verify(PAGE_TITLE, "My Woodworking Forum");

RESTART_TRANSACTION_BOTTOM(); // do not modify this statement
Clear ( TRANSACTION );
END_TRANSACTION();

DO_FreeHttp();

EXIT();
return(0);
```

Sample: Modified WWW Script for Moving the Transaction Loop Statements

The following sample shows the script modified to move the transaction loop statements. The sections in italics are the requests involved with the Logon and Logout. The new locations of the transaction statements are shown in bold.

Sample Script

```
SYNCHRONIZE();


//---------  REQUEST # 1  ---------
//
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 1 - ");

Navigate_To("http://www.myforum.org/");
Verify(PAGE_TITLE, "My woodworking forum - powered by vBulletin");

DO_SLEEP(9);


//---------  REQUEST # 2  (see action item on Page 1)  ---------
//
//  current page url is http://www.myforum.org/
//
// The submit button for this form could not be positively identified so
// an appropriate action has been added (Post_To or Navigate_To).  The
// values for the form fields will be set using Set( ) statement.
//
Set(NEXT_REQUEST_ONLY, POST_DATA, "vb_login_username", "John Smith");
Set(NEXT_REQUEST_ONLY, POST_DATA, "cookieuser", "1");
Set(NEXT_REQUEST_ONLY, POST_DATA, "vb_login_password", "");
Set (NEXT_REQUEST_ONLY, POST_DATA, "s", "3669625a913735e99e31ec9a866a2f28");
Set (NEXT_REQUEST_ONLY, POST_DATA, "do", "login");
Set (NEXT_REQUEST_ONLY, POST_DATA, "forceredirect", "1");
Set (NEXT_REQUEST_ONLY, APPEND_CRLF_AFTER_POST_BODY, TRUE);
Set (NEXT_REQUEST_ONLY, POST_DATA, "vb_login_md5password",
       "df0349ce110b69f03b4def8012ae4970");
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 2 - ");

Post_To("http://www.myforum.org/login.php");
Verify(PAGE_TITLE, "My Woodworking Forum");

BEGIN_TRANSACTION();
RESTART_TRANSACTION_TOP(); // do not modify this statement
DO_SLEEP(3);
```

```
//---------  REQUEST # 4  (see action item on Page 3)  ---------
//
//  current page url is http://www.myforum.org/
//
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 4 - ");

Click_On(LINK, 1, DESCRIPTION, "Tables and Chairs");
Verify(PAGE_TITLE, "My Woodworking Forum – Tables and Chairs");

DO_SLEEP(2);
```

All additional Requests that will occur every transaction

```
RESTART_TRANSACTION_BOTTOM(); // do not modify this statement
Clear ( TRANSACTION );
END_TRANSACTION();
//---------  REQUEST # 5  (see action item on Page 4)  ---------
//
//  current page url is http://www.myforum.org/forumdisplay.php?f=17
//
Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 5 - ");

Click_On(LINK, 1, DESCRIPTION, "Log Out");
Verify(PAGE_TITLE, "My Woodworking Forum");

DO_FreeHttp();

EXIT();
return(0);
```

Preventing Unwanted Subrequests

Overview: Preventing Unwanted Subrequests

Pages your script requests may make a subrequest that causes the transaction to abort. There also may be subrequests that skew the load test results. For example, one of the pages you visit may bring up random advertisements, some of which can take a long time to load. To prevent these subrequests, you can insert one or more Traffic Filter commands.

To modify the script:

1. Identify the URL of the offending subrequest.
2. Insert a Set command with the BLOCK_TRAFFIC_FROM keyword and a string that identifies the URLs to block.

Typically, you place this statement near the top of the script before the BEGIN_TRANSACTION statement, since you always want it in effect. For example, the following command prevents any subrequests to URLs containing "AcmeAds":

```
Set (EVERY_REQUEST, BLOCK_TRAFFIC_FROM, "AcmeAds");
```

Note: The filter string does not need to be a complete URL.

You can also filter the traffic by specifying a substring that must be present in the requested URL. For example, if you are concerned with making subrequests only to pages on your company's web site, you can insert a ONLY_ALLOW_TRAFFIC_FROM traffic filter command. The following example only allows subrequests whose URLs contain the substring "myCorp":

```
Set (EVERY_REQUEST, ONLY_ALLOW_TRAFFIC_FROM, "myCorp");
```

Visual Navigator

If you are using Visual Navigator, you can specify filters by highlighting the Traffic Filters tree item underneath the Playback Options at the top of the tree and type the commands in the top pane of the window.

Conclusion

By following these scripting techniques, you can modify a script to prevent unwanted subrequests. The sample script illustrates the modifications required in the script.

Sample: Preventing Unwanted Subrequests

The following sample uses a filter string to prevent the unwanted subrequest. Points of interest are highlighted in bold.

Sample Script

Set (EVERY_REQUEST, BLOCK_TRAFFIC_FROM, "AcmeAds");

```
SYNCHRONIZE();


BEGIN_TRANSACTION();

RESTART_TRANSACTION_TOP(); // do not modify this statement


//---------  REQUEST # 1  ---------

//

Set (NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 1 - ");


Navigate_To("http://www.mystore.com/");

Verify(PAGE_TITLE, "Jack's Hardware Store");

etc.
```

NetLoad

Using NetLoad

NetLoad is QALoad's suite of load generation scripts that allows you to simulate load conditions on your network using any of the following protocols:

- FTP
- HTTP
- PING
- LDAP
- POP3
- SMTP
- TCP
- UDP
- MSExchange

NetLoad includes QALoad-provided scripts, which you can access from the Conductor to run in a test, for each protocol. You can customize the activity of the script by creating reusable datapools in the QALoad Script Development Workbench to use during testing. When you run a test, each virtual user requests a single datapool record. Once all the records have been read, the datapool file is rewound and the process starts again. You can use QALoad's components to run scripts and analyze the results as usual, or you can integrate your results with Compuware's ServerVantage product.

In short, NetLoad allows you to generate traffic on your network in a controlled manner and gather performance timings from the network. To facilitate testing under TCP/IP and UDP, NetLoad provides you with a server module to simulate server activity — allowing you to gather network timings without expending your actual server resources.

Note: To use NetLoad for MSExchange to test on Outlook 2000, you must ensure that CDO support is installed on your workstation before you continue. For instructions, see Verifying CDO Support for MSExchange.

For more information on the NetLoad Server modules, see NetLoad Server Modules for TCP/IP and UDP.

NetLoad server modules for TCP/IP and UDP

If you are load testing a network running TCP/IP or UDP, you should use the appropriate NetLoad Server module to simulate server responses during your load test. This allows you to load your network and collect timings without expending your own server's resources. The NetLoad Server modules are only for use if you're testing on TCP/IP or UDP. You do not need to install the Server modules to test any other NetLoad-supported protocol.

You can install or copy the NetLoad Server modules to any Windows workstation on your network. After starting the appropriate Server module, you supply the QALoad Script Development Workbench with the host name of the machine where the Server module is running and the port number that you specified when you started the Server module. When you are ready to run a test, start the Server module first. During the test NetLoad communicates with the NetLoad Server module, effectively loading the network. If NetLoad does not find the NetLoad Server module at the specified port—for instance if you mistyped the port number—the test fails (TCP) or fails to initiate (UDP).

Determining when to use the TCP server module

If you are going to send TCP packets using NetLoad, you must have a QALoad TCP Server module running on each machine that you are sending packets to. Copy the TCP Server module file, NetloadTCPServer.exe, to each machine that will be receiving packets and double-click on the file to start the TCP Server module.

Because the QALoad TCP Server module is a Windows-based program, you cannot use it to send NetLoad TCP packets to a UNIX machine.

Determining when to use the UDP server module

It is not necessary to have a QALoad UDP Server module running at the destination machine for NetLoad to successfully send packets to it; however, the Netload UDP Server can be useful to verify that the packets are being sent. To install the UDP Server module on a machine you are sending packets to, copy the program NetloadUDPServer.exe to that machine. Double-click the file to start the UDP Server module.

Since it is not necessary to have the UDP Server module running, you can send NetLoad UDP packets to both UNIX and Windows workstations.

Note: If you are testing UDP in “broadcast” mode, it is not necessary to use the NetLoad Server module.

Installing the NetLoad Server module

If you are load testing a network running TCP/IP or UDP, the NetLoad Server module appropriate for your protocol must be running on a Windows workstation on your network before you start the test. The Server modules are installed automatically if you chose the option to install them during setup. However, once the Server module is installed on one workstation, you can install it on another workstation by simply copying the program from one workstation to another. The NetLoad Server modules are installed to the directory `\Program Files\Compuware\QALoad\Middlewares\NetLoad\Server`, and are named:

- **NetLoadTCPServer.exe** (for TCP/IP): If you are going to send TCP packets using NetLoad, you must have a TCP Server module running on each machine you are sending packets to. Because the TCP Server module is a Windows program, you cannot send NetLoad TCP packets to a UNIX machine.
- **NetLoadUDPServer.exe** (for UDP): It is not necessary to have a UDP Server module running on the machines you are sending UDP packets to. However, the UDP Server is useful for verifying that the packets are being sent. Since it is not necessary to have a UDP Server module installed on the destination workstations, you can send NetLoad UDP packets to UNIX machines.

Starting the NetLoad server module

You can configure and start the NetLoad server module from the Start menu.

If you are load testing a network running TCP/IP or UDP, the NetLoad Server module appropriate for your protocol should be running on a Windows workstation on your network before you start the test. The Server modules are installed with your QALoad product if you chose to install them during setup. If you are unsure if you should be using a NetLoad Server module, see NetLoad server modules for TCP/IP and UDP.

To start the module:

1. Point to **Start>Programs>Compuware> *QA*Load >NetLoad**. Then click on the appropriate Server module: **TCP Server** or **UDP Server**.
2. When prompted, type the port number of the host machine and click **OK**.
3. On the *QA*Load NetLoad Server window, under the **Options** menu, select one of the following:

- Show Message Every Packet — Displays a message, including byte size, after sending or receiving a packet.
- Show Message Every 100 Packets — Displays a message every 100 packets listing the total number of packets received.

Starting a NetLoad session

You can start a NetLoad session from the workbench with an existing datapool file or a new one.

To start a session:

1. From the *QA*Load Script Development Workbench, choose **Session>NetLoad**.
2. Open an existing protocol datapool file or create a new one:
   - To create a new datapool file, choose File>New. The New NetLoad File dialog box opens.
   - To open an existing datapool file, choose File>Open. The Open NetLoad File dialog box opens.
3. Select the protocol you wish to test on and click **OK**. If you are opening an existing datapool file, navigate to the file and open it.
4. Enter or edit the appropriate datapool information in the **Workbook Pane**.

The QALoad Script Development Workbench allows you to have multiple files open at the same time. Datapool files are located in the directory `\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts`.

Creating a NetLoad datapool

To create a NetLoad datapool:

1. From the *QA*Load Script Development Workbench, click **Session>NetLoad**.
2. Click **File>New** to open the New NetLoad File dialog box.
3. Select the protocol for which you wish to create a datapool file and click **OK**.

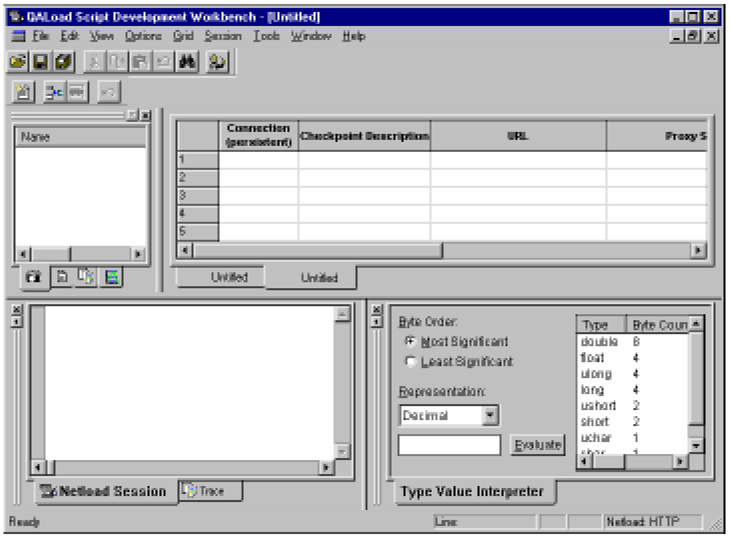
   A grid opens in the Workbook Pane. Each row on the grid represents a single data record. The column headings indicate the appropriate field information to enter. Note that the actual fields in the grid vary by protocol.
4. Enter the appropriate information for your datapool file.

   Some fields on the grid contain pull-down menus. To activate them, click anywhere within the field. Then make your selection from the menu that appears.
5. When you are finished, select **File>Save** to name and save the datapool file.

The datapool file is listed in the Workspace Pane Datapools tab. QALoad creates a script with the same name and lists it on the Scripts tab. Both files are saved to the `\NetLoad\Scripts` directory (for example, `c:\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts\datapool.dat`).

To enter datapool data:

1. From the *QA*Load Script Development Workbench, choose **Session>NetLoad**.
2. Click **File>New** to open the **New NetLoad File** dialog box. Select the protocol for which you wish to create a datapool file and click **OK**. A grid similar to the one shown below appears in the Workbook Pane. Each row on the grid represents a single data record. The column headings indicate the appropriate field information to enter. Note that the actual fields in the grid vary by protocol.



3. Enter the appropriate information for your datapool file. Note that some fields on the grid contain pull-down menus. To activate them, click anywhere within the field. Then make your selection from the menu that appears.
4. When you are finished, click **File>Save** to name and save the datapool file. Note that your datapool file is listed in the **Workspace Pane Datapools** tab. *QA*Load creates a C-based script by the same name and lists it in the Workspace Pane Scripts tab. Both files will be saved to your `\NetLoad\Scripts` directory (for example, `c:\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts\datapool.dat`).
5. (Optional) Write a description of this datapool file for later reference by selecting **Options>NetLoad**. Once a description has been entered for a datapool file, you can review or edit the description any time the file is open by selecting **Options>NetLoad** again.

Editing a NetLoad datapool

You can edit the NetLoad datapool to make changes or additions to the file.

To edit a datapool:

1. With the appropriate NetLoad protocol session open, open the datapool by choosing **File>Open** and navigating to it, or select it from the **Workspace** tab **Datapools** tab.
2. Make any changes or additions to the file.
3. To delete an entire record (a single row), click its row number and select **Grid>Delete Row(s)**.
4. To insert a new record (a single row) above an existing record, click a row number and select **Grid>Insert Row**. NetLoad inserts a blank row above the selected row.
5. Save any changes to the file by selecting **File>Save**.

Adding or editing a NetLoad datapool description

You can add a meaningful description, or edit a previous one, for any NetLoad datapool.

To edit a description:

1. With a datapool file open, select **Options>NetLoad**.
2. Enter a description for the current datapool file.

Datapool fields

The following protocol-specific fields are provided within a datapool:

MSExchange

Checkpoint Description: A description of this checkpoint.

Profile Name: Type the name of your mail profile. For example, Microsoft Outlook.

Send To: Type the names of one or more mail recipients, separated by commas (,) or semi-colons (;).

Cc: Type the names of one or more mail recipients, separated by commas (,) or semi-colons (;).

Size of Body: Select a file size from the drop-down list for the body of the mail message.

Attached file size: Select a file size for the attachment file from the drop-down list.

FTP

Send/Receive: Specifies whether the script will be sending or receiving a file.

ASCII/Binary: Describes whether the file contains ASCII or binary data.

Checkpoint Description: A description of this checkpoint.

Host: The name of the host computer.

User ID: A user ID for accessing the host computer.

Password: A password for accessing the host computer.

File Size Options: Describes whether the file being sent to the host is of fixed or random size.

File Size (min): The minimum file size to send to the host or the size of the fixed file.

File Size (max): The maximum file size to send to the host.

Path: The path of the file to receive, or the destination of the file being sent. You must enter an absolute path.

Filename: The name of the file to receive or of the file being sent.

HTTP

Connection: Describes whether the connection is regular (the connection is closed after the request/response completes) or persistent (the connection remains open for subsequent requests).

Checkpoint Description: A description of this checkpoint.

URL: The address of the page to receive.

Proxy Server: The name of the proxy server (optional). Note that only proxy servers that do not require a user ID and password are supported.

PING

Checkpoint Description: A description of this checkpoint.

Host Name: The name of the host computer.

Pkt Size (Fixed/Random): Describes if the packet being sent to the host is of fixed or random size.

Pkt Size (min): The minimum packet size to send, or the size of the fixed packet to send.

Pkt Size (max): The maximum packet size to send.

LDAP

Checkpoint Description: A description of this checkpoint.

Host Name: The name of the host computer.

Search String: The text string to search for.

POP3

Checkpoint Description: A description of this checkpoint.

POP3 Server: The name of the POP3 server machine.

User ID: A user ID for accessing the POP3 server.

Password: A password for accessing the POP3 server.

Delete after read: Choose whether to delete the message after it has been read.

Connection: Describes whether the connection is regular (the connection is closed after the request/response completes) or persistent (the connection remains open for subsequent requests).

SMTP

Checkpoint Description: A description of this checkpoint.

SMTP Server: The name of the SMTP server machine.

From: Enter an email address or name.

Send To: Type the names of one or more mail recipients, separated by commas (,) or semi-colons (;).

Cc: Type the names of one or more mail recipients, separated by commas (,) or semi-colons (;).

Size of Body: Select a file size from the drop-down list for the body of the mail message.

File Path: Select a file from the drop-down list to use as the body of the mail message. This field displays files in the local directory only if you selected Browse in the Size of Body field.

Attached file size: Select a file size for the attachment file from the drop-down list.

Attached file path: Select a file from the drop-down list to use as an attachment. This field displays files in the local directory only if you selected Browse in the Attached File Size field.

Connection: Describes whether the connection is regular (the connection is closed after the request/response completes) or persistent (the connection remains open for subsequent requests).

TCP

Checkpoint Description: A description of this checkpoint.

Host Name: The name of the host computer.

Port: The port number of the host computer.

Pkt Size (Fixed/Random): Describes if the packet being sent to the host is of fixed or random size.

Pkt Size (min): The minimum packet size to send, or the size of the fixed packet to send.

Pkt Size (max): The maximum packet size to send.

UDP

Checkpoint Description: A description of this checkpoint.

Host Name: Type the name of the host computer that is to receive the packet.

Port: The port number of the host computer.

Pkt Size (Fixed/Random): Describes if the packet being sent to the host is of fixed or random size.

Pkt Size (min): The minimum packet size to send, or the size of the fixed packet to send.

Pkt Size (max): The maximum packet size to send.

Verifying CDO support for MSExchange

Before you can successfully test with NetLoad for MSExchange using Outlook 2000, you must ensure that Collaboration Data Objects (CDO) support is installed.

To verify CDO support:

1. From the Windows task bar, click **Start>Settings>Control Panel**.
2. Double-click the **Add/Remove Programs** icon.
3. From the list on the **Install/Uninstall** tab, select Microsoft Office 2000 or Microsoft Outlook 2000.
4. Click the **Add/Remove** button.
5. Click **Add or Remove Features**.
6. Click the plus sign (**+**) next to Microsoft Outlook for Windows.
7. Select **Collaboration Data Objects**, and then click **Run from My Computer**.

UNIX

Installing UNIX Players

The necessary UNIX Player software is distributed with your QALoad Windows installation.

Note: For updated UNIX system requirements and installation procedures, please refer to the QACenter Performance Edition Installation and Configuration Guide. You can access this guide by clicking Start>Programs>Compuware>QALoad>Documentation>Installation and Configuration Guide.

Transferring scripts to a UNIX Player

Normally, the appropriate script is automatically uploaded from the QALoad Conductor to the Players and compiled at runtime. However, if it is ever necessary to manually transfer a script, use the procedure that follows.

Note: The machine where the QALoad Script Development Workbench is installed must have Winsock-based TCP/IP to transfer a script to the UNIX machine where you wish to run it.

To transfer a script:

The following procedure describes how to transfer a script file from the Windows workstation where the QALoad Script Development Workbench resides to the system running the QALoad Player.

1. Access the Script Development Workbench.
2. From the **Session** menu, choose the middleware session you want to start.
3. In the **Workspace Pane**, click the **Scripts** tab.
4. On the **Scripts** tab, select the script you want to transfer.
5. From the **Tools** menu, choose **FTP** to open the FTP Transfer dialog box. Note that the file name you selected to transfer appears in the **File to Transfer** field.
6. Enter the **Host Name**, **User Name**, **Password**, and **Destination Directory**.
7. Click **Transfer** to send the file to the system where your *QA*Load Player is installed.
8. If you want to save the information you have entered for subsequent transfers, click **Save Settings**.
9. Click **Close/Abort** to exit the FTP Transfer dialog box.

Testing with QARun

Creating a QARun script

To create a QARun script, insert any number of QARun transactions (QARun scripts) into a QALoad template script accessible from the QALoad Script Development Workbench. The template script is a simple QALoad script that can be compiled and run; however, it contains no functionality until you insert the QARun transactions appropriate for your testing needs. QALoad provides two methods for inserting QARun transactions: automated and manual.

Using the automated method, you enter information in the QALoad Script Development Workbench about the QARun transactions to use and then let QALoad generate the test script using the information you provided. This method is fast and efficient when you know exactly which QARun scripts to use and where they are located.

The manual method allows you to open a copy of the QALoad template script and insert transactions and commands manually. You may want to use this method if you suspect you may need to edit your script while you're creating it.

Automatically creating a QARun script

To automatically create a QARun script:

1. From the *QA*Load Script Development Workbench, click **Session>QARun** to start a *QA*Run scripting session.
2. Click **Session>Generate Script**. The Create New QARun Execution Script dialog box opens.
3. In the **Login String** field, select or type a valid username and password to access your installation of *QA*Run.
4. In the **Environment** field, select the appropriate *QA*Run environment.

5. In the **QARun Script** field, enter the name of the *QA*Run transaction to insert, or select it from the list, which contains a record of the last five *QA*Run script names you entered.

   Although you can enter a script name from any database, when the test is actually running and *QA*Load invokes *QA*Run, *QA*Run attempts to retrieve that script from its default database. Therefore, in the *QA*Run program installed on the Player, you should designate a default database that contains the script(s) you want to run.
6. Select the **Automatically Include Checkpoint** check box if you want *QA*Load to automatically insert a checkpoint into the script after this *QA*Run transaction.
7. In the ***QA*Load Script Name** field, enter a name for this *QA*Load script. To write over an existing script, click the **Browse** button to the right of this field and select a script from the list of available scripts.
8. To add additional *QA*Run transactions to this script, click **Add Script** and repeat Steps 3–6 for each additional transaction.
9. When you are finished, click **Create Script**. The *QA*Load script is saved in the directory `\Program Files\Compuware\QALoad\Middlewares\QARun\Scripts`, and the script opens in the script editor.
10. To compile the script for testing, click **Session>Compile**.

Manually creating a QARun script

You can manually insert QARun commands or scripts into a QALoad script to compile.

To manually create a script:

1. From the *QA*Load Script Development Workbench, select **Session>QARun** to start a *QA*Run scripting session.
2. Select **Session>New Template** to create a new script from the *QA*Load template script.
3. In the Choose Script Name dialog box, enter a name for the new *QA*Load script in the **Script Name** field and click **OK**. The script is saved in the directory `\Program Files\Compuware\QALoad\Middlewares\QARun\Scripts`, and the script opens in the script editor.
4. Edit the script as necessary:
   - You can manually enter any transactions or scripting commands directly in the script.
   - You can insert a *QA*Run transaction by positioning the cursor on the appropriate line and selecting **Session>Insert>Transaction**. On the **Insert a QARun Transaction** dialog box that opens:
     - In **Login String**, select or type a valid user name and password to access your installation of *QA*Run.
     - In **Environment**, select the appropriate *QA*Run environment.
     - In **QARun Script Name**, enter the name of the *QA*Run transaction to insert, or select it from the list, which contains a record of the last five *QA*Run script names you entered. Note that you can enter a script name from any database; however, when the test is actually running and *QA*Load invokes QARun, *QA*Run will attempt to retrieve that script from its default database. Therefore, in the *QA*Run program installed on the Player, you should designate a default database that contains the script(s) you want to run.
     - When you are finished, click **Insert** to insert the script you just created into the *QA*Load script.
5. When you are finished, save any changes.
6. To compile the script for testing, select **Session>Compile**.

Troubleshooting

The Default Session Prompt didn't open?

If the Default Session Prompt fails to open when you start a middleware session, then default session checking was previously disabled. Do the following to enable default session checking:

1. From the **Options** menu, choose **Workbench**. The Configure *QA*Load Script Development Workbench dialog box opens.
2. On the **Workbench Configuration** tab, select the **Enable default Session checking** check box.

The next time you open a QALoad Script Development Workbench middleware session, you are prompted to make it your default session.

Winsock running out of socket resources

You may encounter a problem running out of socket resources on NT or Solaris when there are large numbers of short-lived connections.

When TCP/IP connections are shutdown, they go into a TIME_WAIT state waiting for the specified interval to expire. While in that state the connection is looking for any stray packets that may have been sent to this connection and remain unacknowledged.

If this process was skipped, it would be possible for a new connection to be opened using the same address and port as the previous connection and to incorrectly receive data that was intended for the previous connection. When QALoad is generating many short-lived connections, during a Winsock or WWW load test, the default setting for the timed wait delay may be so high that the driver machine will run out of socket resources as all closed sockets wait in the TIME_WAIT state.

To change the setting for the timed wait delay:

Windows NT

Set the registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\
Parameters\TcpTimedWaitDelay
```

to a lower value. It can be set to anything between 30 and 300. Compuware suggests using the lowest possible value (30).

Solaris 2.6

Using "ndd" set the "tcp_close_wait_interval" to 30 seconds:

```
ndd -get /dev/tcp tcp_close_wait_interval
```

```
ndd -set /dev/tcp tcp_close_wait_interval 30000
```

Solaris 2.7

Use "ndd" as shown previously for Solaris 2.6, but substitute the "tcp_time_wait_interval"

SAP script validation fails

If your SAP script fails during validation, you may need to disable automatic proxy configuration in Internet Explorer.

To disable automatic proxy configuration:

1. In Internet Explorer, click **Tools>Internet Options**.
2. On the **Connections** tab, click the **LAN Settings** button.
3. Ensure that the **Use automatic configuration script** check box is cleared.

If disabling the automatic proxy configuration does not solve the problem, consider increasing the script execution timeout value to 100 seconds or to the length of the capture file (in seconds), whichever is greater.

To increase the timeout value:

1. With an SAP session open in the Script Development Workbench, click **Options>Workbench**.
2. On the **Script Validation** tab, type the new value in the **Wait up to** field.
3. Click **OK**.

Linking errors during validation or compilation of SAP scripts

When you re-record SAP 4.x scripts for SAP 6.20/SAP 6.40, you must click the Build SAP Libraries button on the SAP Convert Options dialog box. This button generates new libraries based on the version of SAP that is currently installed. If you have upgraded to a newer version of SAP and do not update the libraries, you may experience various linking errors during validation or compilation.

Performance issues with SAP or Citrix scripts

If you experience performance issues with SAP or Citrix scripts, increase your system paging file size to a fixed size of at least four times the amount of RAM on the machine.

Recording Citrix Scripts for Restricted Desktops

When recording a Citrix script on a restricted desktop, you must take extra steps to ensure proper playback. Dynamic windows may be created and destroyed with no user interaction at both logon and logoff. To prevent unexpected results during playback, do not click on any of these dynamic windows. For example, the script might attempt to click on a destroyed window or a window that has not been created yet. The appearance of dynamic windows often depends on the speed of the server or the load on the farm during playback.

Desktop Screen Resolution When Recording Citrix Scripts

To ensure that the entire Citrix interface is visible during recording, set the Resolution field in the Citrix recording options to a lower value than that of the desktop. Also, the screen resolution must be the same as the screen resolution specified in the Citrix ICA file.

Wait Points in Citrix Scripts

When mouse move consolidation is enabled via the Combine Mouse Input option on the Convert Options dialog box, not all events that were captured appear during replay. As a result, the placement of wait points for window moves and window resizing is important. When a window is moved on the desktop, a window move event is created for each mouse move. However, during replay of a consolidated script, only one mouse move is made (to the final destination), and subsequently only one window move event occurs. Because of this consolidation of consecutive mouse and window moves, the `WaitForWindowMove` command in your script should target the final window move in the series. This issue also applies to the `WaitForWindowResize` command and the resizing of windows on the desktop.

If a window's title changes while the window is being created (such as a browser window's title bar changing once the default Web page begins to load), the `WaitForWindowCreate` command may time out. If this occurs, remove the wait point or insert a `WaitForScreenUpdate` command to preserve the wait in the script.

Analyze

Overview of QALoad Analyze

QALoad Analyze is the QALoad component used to create summary statistics and graphs from timing data collected during a load test. Set criteria for collecting and displaying test data in QALoad Analyze before or after opening a test's timing file (.tim). For example, alter output options, time ranges, and graphics display options.

QALoad Analyze stores the state of a timing file when it is closed so that next time you open it, you see the same reports and graphs that were present last time you viewed it. You also have the ability to easily create templates, which enable you to specify the reports and graphs that are automatically generated for any new timing file you open.

In addition, QALoad Analyze generates a working folder where all files and reports related to the timing file are stored. QALoad Analyze provides seven pre-defined reports as well as the ability to create custom reports using XML file (.xml), XSL translation file (.xsl), and HTM file (.htm) formats. View these reports in QALoad Analyze or in a Web browser.

QALoad Analyze displays a timing file tab in the Workspace, each tab containing groups. Use QALoad Analyze's interactive view to sort test data, produce detailed checkpoint data, produce a variety of graphs and reports (with drag and drop functionality), export data to different formats, and email test results and pre-defined reports.

Understanding durations

When you begin to analyze your test results, it is important to understand how durations are calculated by QALoad.

Transaction duration

Transaction duration is the time that the script being tested takes to complete a transaction, from the `BEGIN_TRANSACTION` command to the `END_TRANSACTION` command.

Three factors comprise transaction duration:

- The script processing time including, but not limited to, added script logic, *QA*Load processing of server replies, and other *QA*Load processing.
- Sleep time.
- The response time of the application under test including, but not limited to, the application server, database access, and network.

Checkpoint duration

Checkpoint duration is the amount of time between begin and end checkpoint statements. The following factors comprise checkpoint duration and apply to both automatic checkpoints and user-defined checkpoints.

If you select the Conductor's Enable timing of automatic middleware checkpoints option or use the `BeginCheckpoint` and `EndCheckpoint` functions in the script, the following factors comprise checkpoint duration:

- The response time of the application under test, including, but not limited to, the application server, database access, and network.
- Sleep time, if the Conductor's **Include sleep times when calculating checkpoint timings** option is selected.
- *QA*Load processing time is not included within these checkpoints in order to provide a more accurate value of server, database, and network response times.

Checkpoint durations do not always sum to the same value as the transaction duration. For more information, see Comparing checkpoint durations to transaction duration.

QALoad Analyze Menus

QALoad Analyze menus and toolbar buttons

Click a menu or toolbar name in the following list for a description.

File

Edit

View

Templates

Tools

Window

Help

Analyze Toolbar buttons

Graph Toolbar buttons

File menu

Item	Description
Open	Opens a dialog box to select a timing file for Analyze to process. When a timing file is opened, Analyze opens a Summary report in the Data window and corresponding timing file tab in the Workspace.
Close	Closes the current timing file view displayed in the Data window. In the Workspace, the associated group and timing file tab will remain open.
Close Timing File	Closes the current timing file and all reports, detail views, or graphs associated with that timing file.
Print	Opens the Print dialog box to set options for printing the current Summary report.

Print Preview	Opens a window displaying the current Summary report as it will look when printed.
Print Setup	Opens the Print Setup dialog box to select a printer and to set options for printing.
Export	Opens the Save As dialog box to save a load test Detail view as a CSV (*.csv) file, HTML (*.htm, *.html) file, or save graphs to HTML (*.htm, *.html) files.
Send	Opens the Send dialog box, which enables you to save test data outside Analyze, package test data into .zip files, or email test data.
Properties	Opens the Properties dialog box, which displays details about Analyze and the current timing file. This dialog box also contains buttons that enable you to save the displayed information to the clipboard or in a file.
Exit	Exits the Analyze program.

Edit menu commands

Option	Description
Copy	Copies selected text to the clipboard.
Select All	Selects all counters, checkpoints, data points, etc., of the active timing file in the Workspace.
Unselect All	Un-selects all previously selected counters, checkpoints, data points, etc., in the Workspace of the active timing file.

View menu commands

Option	Description
Toolbar	Displays or removes the toolbar from

	the top of the screen.
Status Bar	Displays or removes the status bar from the bottom of the screen.
Detail	Opens a Detail information view in the Data Window for the selected checkpoints.
Graph	Opens the Select Graph dialog box to choose a graphical format for reviewing the current data.
as Web Page	Opens the current report, graph, or detail view, in the default Web browser.
Workspace	Displays or hides the Workspace.
Workbook	Displays or hides the Data window.

Template Menu Commands

Option	Description
Save Current Views...	Opens the Specify Template to Save dialog box that allows you to name the new template file. The default is the SessionID portion of the file name of the open timing file.
Use Existing...	Opens the Select Template to Use for Open Timing Files dialog box, where you can select the template to apply to your open timing files. This closes all current reports and graphs and opens the views specified in the template.
Remove...	Opens the Select Template(s) to Remove From Disk dialog box, where you can remove stored templates from disk. No open views are affected by this action and no templates are checked for dependencies.
Thresholds...	Opens the Thresholds dialog box where you can create or edit system-wide thresholds.

Tools menu commands

Option	Description
Options	Provides access to options for customizing data.
Show Replay	Launches the Conductor to replay a test recording.
Conductor	Starts QALoad's Conductor program, which is used to control all testing activity.
Workbench	Starts the QALoad Script Development Workbench, which is used to create and manage test scripts.

Window menu commands

Option	Description
Cascade	Automatically moves and resizes all the active windows, so they overlap one another.
Tile Horizontally	Moves and resizes all the active windows so they are lined up horizontally.
Tile Vertically	Moves and resizes all the active windows so they are lined up vertically.
Arrange Icons	Arranges any minimized windows within QALoad Analyze's parent window.
Close All	Closes all open windows.

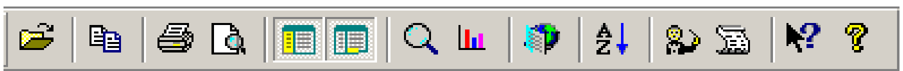
Help menu

Option	Description
Help Topics	Displays QALoad's online help contents.
About QALoad Analyze	Displays the program's About box and copyright notice.

QALoad Analyze Toolbars

Analyze toolbar buttons

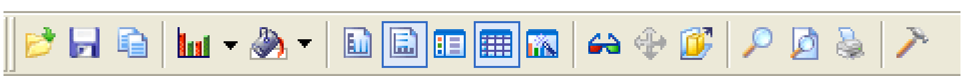
Click a toolbar button for a description of that button.



Graph toolbar buttons

Change the style and appearance of a graph using options available from the Graph toolbar. The Graph toolbar also contains buttons for standard Windows operations. Although it normally appears atop a graph, the toolbar is completely dockable. Move the toolbar to another side of the graph, or off the graph altogether, by clicking any unpopulated area of the toolbar and dragging it to another area.

Click any button in the following toolbar for a description of that button.



Display the Graph toolbar by right-clicking in an open area of a graph and choosing the Toolbar option from the shortcut menu.

Accessing Test Data

Using timing files

When you run a test using a particular session ID file (set up in the Conductor), each Player compiles a local timing file comprised of a series of timing records for each checkpoint of each script run on that Player. Each timing record in the file consists of a response time/elapsed time pair of values specifying the amount of time it took a certain checkpoint to finish (response time) at a specific time in the test (elapsed time).

At the end of a test, Player timing files are sent to the Conductor and are merged into a single timing file, called the Primary timing file, for analysis. If you set up integration with Compuware's ServerVantage product, the Conductor collects timing data from the ServerVantage central console and merges that data into the timing file as well.

Primary timing files are saved in the `\Program Files\Compuware\QALoad\TimingFiles` directory, and are name `<sessionID>_date_time.tim`.

The Primary timing file created by the Conductor after a test run contains all of the timing records of all Players in that test run. Use QALoad Analyze to view, sort, graph, and create reports using the test data in the timing file.

Hint: In the event that something goes wrong on the network and a Player timing file is not passed to the Conductor, it is still possible to analyze results from a Player timing file. Player timing files are saved in the `\Program Files\Compuware\QALoad\TimingFiles` directory and are named **`tim_yyyymmdd_hhmmss_xxx`**`.ptf`, where **`yyyymmdd_hhmmss`** is the date/time the test was started, and **`xxx`** is the Player number.

Accessing test data

When you open a timing file, QALoad 's Analyze program summarizes the checkpoints recorded in the file during the load test and presents the data in a report format called the Summary report.

Three ways to access QALoad Analyze and open a timing file containing test results are:

To access Analyze from the QALoad Conductor:

1. In the *QA*Load Conductor, click **Tools>Options**. The Options dialog box appears.
2. Click the **General** tab. In the General Options area, select the **Launch Analyze After Test** check box.

At the end of each test run, QALoad Conductor automatically launches QALoad Analyze and opens the most recent timing file. Or, if you did not select the Launch Analyze After Test check box before the test:

1. Click **Tools>Analyze**.
2. In *QA*Load Analyze, click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.

Use the following method when accessing a previously-created timing file.

To access Analyze from the Windows Start menu:

1. Click **Start>Program Files>Compuware> *QA*Load >Analyze**.
2. Click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.
3. Select the template to use for viewing the timing file.

Use the following method when you are already working in the QALoad Script Development Workbench and need to access a previously-created timing file.

To access Analyze from the QALoad Script Development Workbench:

1. In the *QA*Load Script Development Workbench, click **Tools>Analyze**.
2. In *QA*Load Analyze, click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.
3. Select the template to use for viewing the timing file.

Accessing Test Data via Groups

Located in the QALoad Analyze Workspace, each group displays data from a timing file. The data displayed and the groups available may vary, depending on the type of data that was collected during the load test.

Access groups to select data for generating a detail view or graph. Click a group name below to view the type of data that is displayed by each group.

Reports

Checkpoints

Counters

Server Monitoring

Player Performance Counters

Top Processes

RIP Files

ApplicationVantage

Expert User

Using Templates

Using Timing File Templates

Timing file templates enable you to save current views of an open timing file. All open views, such as reports, graphs, and thresholds are stored in the template. When you reopen a timing file or open a new timing file and apply a template, it appears with the set of views defined for the template.

Every time you close a timing file in Analyze, a Last Viewed State template for the timing file is created. This file stores the reports and graphs that are open, their positions and sizes, and any thresholds you defined. You can use an option s in the Open menu or in the Tools>Options menu to choose to reopen a timing file in the same state its last viewed state.

Use the Template menu to:

- Save the current views of a timing file to create a template.
- Select an existing template and apply it to the open timing file.
- Remove the template from the current timing file or from disk.
- Create system-wide thresholds for counters and checkpoints.

You can apply a template to any timing file you open. If any part of the template cannot be applied to the timing file, for example, if the template references a script that doesn't exist in the current file, a dialog box displays with the name of the report or graph that doesn't apply. The default name for the template is the SessionID portion of the file name of the open timing file.

Note: Only one timing file is saved to a template. If more than one timing file is open, the option to define a template is disabled.

Creating a New Template

Create a template to save the views of an open timing file. This saves reports and graphs that are open, as well any thresholds that have been defined. When you open other timing files using the template, the files display in the views saved to the template.

To create a new template:

1. Click Template>Save current views. The Specify Template to Save current views and thresholds dialog box appears.

   Note: This menu option is available only if a single timing file is open.

2. Select a folder in the Save in field. By default, all templates are stored in the Templates folder. The default template name is the Session ID portion of the timing file name.
3. Do one of the following:
   - Click Create to accept the Session ID as the template name. When you use this name for the template, all future timing files created by this Conductor session automatically open with the views specified in the template.
   - Type a name for the new template in the File name field, and click Create.

   Note: You can choose template options that globally set how timing files open using the options in the Analyze Options dialog box .
   .

Opening a Timing File in a Template

When you open a timing file, you can select how it displays by opening it in a template. Specifying a template opens the reports, graphs, and thresholds defined in the template.

To open a timing file in a template:

1. Click **File>Open**. The **Open Timing File** dialog box displays.
2. In the **File name** field, select the timing file to open.
3. Select one of the following:

   - Open last viewed state if available - (Default) Opens the timing file in the same views that were displayed when you last closed it. This restores all reports and graphs that were open, their positions and sizes, as well as any defined thresholds for graphs. If there is no last viewed state or if you do not select this option, the file opens using the option you select in the Template area below.
   - In the Template area, click one of the following options:

   o Do NOT use a template when opening timing file - No template is applied and only the Summary report for the timing file displays.

   o Use template associated with Session ID file name - (Default) Applies the template with the same Session ID name as the timing file. Use this option to open all future timing files created by this Conductor session with the views specified in the template. If no match is found, only the Summary report displays.

   o Use this template for opening the timing file - Enables the Browse (...) button. Select a saved template to apply to all timing files when they are opened.

Note: By default, both the Open in last viewed state if available and Use template associated with the Session ID file name are selected. This way if there is no last viewed state, the template associated with the Session ID file name is applied to the timing file.

4. Click **Open**. The timing file you selected appears in the views defined for the selected template.

Applying a Template to an Open Timing File

You can select an existing template and apply it to all open timing file. This closes all the reports and graphs that are open and displays the timing file in the views defined in the template.

To apply a template to an open timing file:

1. Click Template>Use existing. The Select Template to Use for Open Timing Files dialog box appears.
2. Select a template and click Apply.

All open reports and graphs are closed and those specified in the template are opened.

Note: If any part of the template cannot be applied to the timing file, for example, if the template references a script that doesn't exist in the current file, a dialog box appears with the name of the report or graph that can't be displayed.

Applying Templates Globally

You can select template options that globally set how timing files open. Using a template opens the reports, graphs, and thresholds defined in the template.

To select a template option for application to files when they are opened:

1. Click **Tools>Options**. The **Analyze Options** dialog box displays.
2. Click the **Templates** tab.
3. Select one of the following:
   - Open last viewed state if available - (Default) Opens the timing file in the same views that were displayed when you last closed it. This restores all reports and graphs that were open, their positions and sizes, as well as any defined thresholds for graphs. If there is no last viewed state or if you do not select this option, the file opens using the option selected in the Template area below.
   - In the Template area, select one of the following:
     - Do NOT use a template when opening timing files - No template is applied. The Summary report for the timing file displays.
     - Use template associated with the Session ID file name - (Default) Applies the template with the same Session ID name as the timing file. Use this option to open all future timing files created by this Conductor session with the views specified in the template. If no match is found, only the Summary report displays.
     - Use this template for opening timing files - Enables the Browse (...) button. Select a saved template to apply to all timing files when they are opened.

Note: By default, both Open in last viewed state if available and Use template associated with the last Session ID file name are selected. This way, if there is no last viewed state, the template associated with the Session ID file name is applied to the timing file.

4. Click **Ok**.

Using the Options Dialog Box for Templates

Use the Analyze Options dialog box to select a template option that applies when you open all timing files. Using a template opens the reports, graphs, and thresholds defined in the template. Setting the template option here determines how all timing files open.

Note: Use Template>Use Existing to apply a template to an individual open timing file.

To specify a template:

1. Click **Tools>Options.**
2. Click the **Templates** tab and select one of the following:
   - Open last viewed state if available - (Default) Opens the timing file in the same views that were displayed when you last closed it. This restores all reports and graphs that were open, their positions and sizes, as well as any defined thresholds for graphs. If there is no last viewed state or if you do not select this option, the file opens using the option selected in the Template area below.

- In the Template area, select one of the following:
  - Do NOT use a template when opening timing files - No template is applied. The Summary report for the timing file displays.
  - Use template associated with the Session ID file name - (Default) Applies the template with the same Session ID name as the timing file. Use this option to open all future timing files created by this Conductor session with the views specified in the template. If no match is found, only the Summary report displays.
  - Use this template for opening timing files - Enables the Browse (...) button. Select a saved template to apply to all timing files when they are opened.

Note: By default, both Open in last viewed state if available and Use template associated with the Session ID file name are selected. This way if there is no last viewed state, the template associated with the Session ID file name is applied to the timing file.

Displaying Detail Data

Detail Views

You can view detail data by right-clicking a script or group and selecting Detail. The detail view displays data from a timing file. The data displayed and the groups available may vary, depending on the type of data that was collected during the load test. Detail data is displayed in two panes: a summary table and a data table.

You can view details for the following groups:

Checkpoints

Counters

Server Monitoring

Player Performance Counters

Top Processes

Expert Users

Displaying detail data

Display detailed statistics from a timing file such as checkpoints, counters, in the QALoad Analyze Data window. View statistics for not only the active timing file, but also for other timing files and drag and drop onto the active timing file detail view.

To display detailed statistics:

1. In the workspace, with the appropriate Timing File tab selected, click the group for which you want to view statistics.
2. Select the appropriate checkpoints or counters (depending on which group you choose).
3. From the Analyze toolbar, click the **Detail** button or right-click on a selected checkpoint or counter and choose **Detail**.

Detail information is presented in the Data window in both a summary and data table. The information displayed varies based on the group selected.

Note: If the test aborts, complete data for all the checkpoints and counters may not display.

The following detail views are available:

Checkpoints Detail Data

Counters Detail Data

Server Monitoring Detail Data

Player Performance Counters Detail Data

Top Processes Detail Data

Expert user Detail Data

Checkpoints Detail Data

You can view checkpoint detail data by Script or by Player. The View by Script displays the detail data of selected checkpoints and raw data collected for each script during the load test. The View by Player groups the detail data by player machine and shows only the scripts for which individual players gathered data. For each view, checkpoint detail data is displayed in two panes: a summary table and a data table.

Checkpoints Summary Table

Shows statistical averages for the selected checkpoints and displays a summary of the raw data collected from the load test. The following data may be displayed:

Timing File: Name of the timing file the checkpoint came from.

Script: Name of the script file.

Checkpoint: Checkpoint name

Type: Type of checkpoint: response time or sleep time.

Group: Automatic (available if automatic checkpoint timings are enabled in the Conductor), or User (for user-defined checkpoints).

#Trans: Total number of data points that were used to calculate the statistics. If data thinning is enabled, this column displays as "#Thinned Trans".

#Recs: Number of records recording during the test. If data thinning is enabled, this column displays as "#Thinned Recs".

Data Thin: If the Enable Timing Data Thinning check box was selected in the QALoad Conductor's Timing Options dialog box prior to starting a load test, the value typed in the Thin Every <xx> Transactions will be in this column. If not selected, the value is none.

Note: For a complete description of this QALoad Conductor option, see Timing Options.

Min: Minimum recorded response time.

Mean: Average of the response times.

Max: Maximum recorded response time.

StdDev: Standard deviation of all response times. Standard deviation is an indicator of how widely values are dispersed from the average (mean) value. A large standard deviation indicates a wide variance in response times.

Median: Median response time (in seconds). The median is the value at which half of the responses are greater and half of the responses are less. If the number of responses is large, the median is usually close to the mean.

Nth Percentile: Displays that nth% of the responses have a value less than the value shown.

Pacing (Seconds): Rate at which the script executed transactions.

VU's: Number of virtual users executing this script.

Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Failure Rate: Percentage of checkpoints that surpassed the threshold limit.

Total Violations: Total number of threshold violations.

Checkpoints Data Table

Provides a view of the raw data collected during the load test. It can be useful for pinpointing anomalies with load test results. The following data may be displayed:

Timing File: Name of the timing file the checkpoint came from.

Script: Name of the script file.

Checkpoint: Checkpoint name.

Type: Type of checkpoint: response time or sleep time.

Group: Automatic (available if automatic checkpoint timings are enabled in the Conductor), or User (for user-defined checkpoints).

VU: The virtual user that was running.

Player: Player machine the test results came from.

#Samples: Displays how many records were thinned into a single record if data thinning was enabled. If the value is 1, the data records were not thinned.

Elapsed (Seconds): Time into the test at which a data point was collected.

Response (Seconds): Value of the data collected.

Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Counters Detail Data

Counter detail data is displayed as follows:

Counters Summary Table

Shows statistical averages for the selected counters. It is a summary of raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Script: Name of the script file that contained the counter.

Group: Group name.

> Note: Not all counters will belong to a group. Those counters that do will have a group name displayed. For instance, custom and Web counters are logically organized by groups. However, Virtual Users and Total Virtual Users do not belong to a group.

Name: Name of the counter.

> Note: Certain statistical data is shared across all detail views. For a description of these fields, click the following: Statistical Information

Counters Data Table

Provides a view of the raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Script: Name of the script file that contained the counter.

Group: Group name. Custom and Web counters are logically organized by groups.

Counter:

Instance:

VU: Name of the virtual user.

Player: Player machine the test results came from.

#Recs: Displays how many records were thinned into a single record if data thinning was enabled. If the value is 1, the data records were not thinned.

Elapsed (Seconds): Time into the test at which a data point was collected.

Value: Value of the data collected. This column displays the value of instance counters.

Cumulative Value: Total number of occurrences during the elapsed time. This column displays the value of cumulative counters.

Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Server Monitoring Detail Data

QALoad provides performance counter data through three server monitoring methods. Click the following links for descriptions of the server monitoring detail data options:

- Remote Monitoring - Monitoring of performance counters from a machine under test without the use of agent software on the machine.
- Server Analysis - Monitoring of performance counters from a machine under test using the ServerVantage agent software installed on the machine.
- ServerVantage - Availability Management application complimentary to *QA*Load for service level monitoring of performance counters for applications, servers, and databases during production. ServerVantage also provides notification, event management, and reporting features.

Player Performance Counters Detail Data

Player performance counters detail data is displayed as follows:

Player Performance Summary Table

Shows statistical averages for the selected Player performance counters. It is a summary of the raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Player: Player name.

Description: Description of the counter.

Note: Certain statistical data is shared across all detail views.

Player Performance Data Table

Provides a view of the raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the performance counter came from.

Player: Player name.

Description: Name of the counter.

Elapsed (Seconds): Time into the test at which a data point was collected.

Value (%): Value of the data collected.

Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Top Processes Detail Data

Top Processes detail data is displayed as follows:

Top Processes Summary Table

Shows statistical averages for the selected Top Processes data. It is a summary of the raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the counter came from.
Process: Name of the process monitored during the test.
#DataPts: Number of data points collected for each particular process.

Note: Certain statistical data is shared across all detail views. For a description of these fields, click the following: Statistical Information

Top Processes Data Table

Provides a view of raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the counter came from.
Process: Process name.
Elapsed (Seconds): Time into the test at which a data point was collected.
% CPU: Percent of CPU used.
Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Expert User Detail Data

Expert User detail data is displayed as follows:

Expert User Summary Table:

Shows statistical averages for the selected expert user data. It is a summary of the raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the expert user data came from.

Script: Name of the script the expert user data came from.

Player: Name of the Player machine the expert user data came from.

Main URL: The of the main URL requested.

Response: The HTTP status code of the request returned by the server.

Subrequest URL: Lists the URL of the subrequests that were made from the Main URL.

Data Type: The type of data. This can be Server or Network.

> Note: Certain statistical data is shared across all detail views. For a description of these fields, click the following: Statistical Information

Expert User Data Table

Provides a view of the raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the performance counter came from.

Script: QALoad Player agent name.

Player: Server analysis counter.

Main URL: Multiple occurrences in time of the performance counter.

Response: The HTTP status code of the request returned by the server.

Subrequest URL: Lists the URL of the subrequests that were made from the Main URL.

Data Type: The type of data. This can be Server or Network.

Elapsed (Seconds): Time into the test at which a data point was collected.

Response (Seconds): Network and server response times in the time scale you select in Tools>Options>Workspace. You can select seconds, minutes, or hours. The default is seconds.

Threshold: Value that sets the limit for the threshold, which is the expected warning and critical level for the checkpoint.

Sorting test data

A Detail view potentially contains a large number of checkpoints, counters, and so forth, especially if a load test had many virtual users. To make information manageable, specify up to three levels of criteria to sort by, in ascending or descending order.

For example, if a test ran using five scripts on 100 virtual users, sort the data by script name. Suppose each virtual user ran more than one transaction using a particular script, then sort by both script name and by virtual user. Or, to quickly locate any timing bottlenecks, sort by response time.

Use the Sort Details dialog box to sort a detail view. To access this dialog box, select Tools>Sort from the Analyze menu or click Sort on the Analyze toolbar.

Specify sort options for the following grid: Select the Summary option to conduct sort on the Summary table. Select the Data option to conduct a sort on the Data table.

Sort By: Select the first column to sort by from the list, then select the Ascending or Descending sort order option.

Then By: Select the second column to sort by from the list, then select the Ascending or Descending sort order option.

Then By: Select the third column to sort by from the list, then select the Ascending or Descending sort order option.

Graphing QALoad timing data

A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

Select the group to graph:

In the Workspace, with the appropriate Timing File tab selected, click the group for which to create a graph.

Note: If the test aborts, complete data may not be available for all checkpoints and counters.

The following groups are available, depending on the timing file:

Checkpoints
Counters
Server Monitoring
Player Performance Counters
Top Processes
Expert User

Note: For each Group except Checkpoints and Expert User, the graph type is a line graph. For graphing multiple checkpoints or expert users, the graph type is either a line or bar graph. For graphing a single checkpoint only, in addition to line and bar graphs, you can also create Response Time Distribution and Cumulative Response Time Distribution graphs.

Thresholds

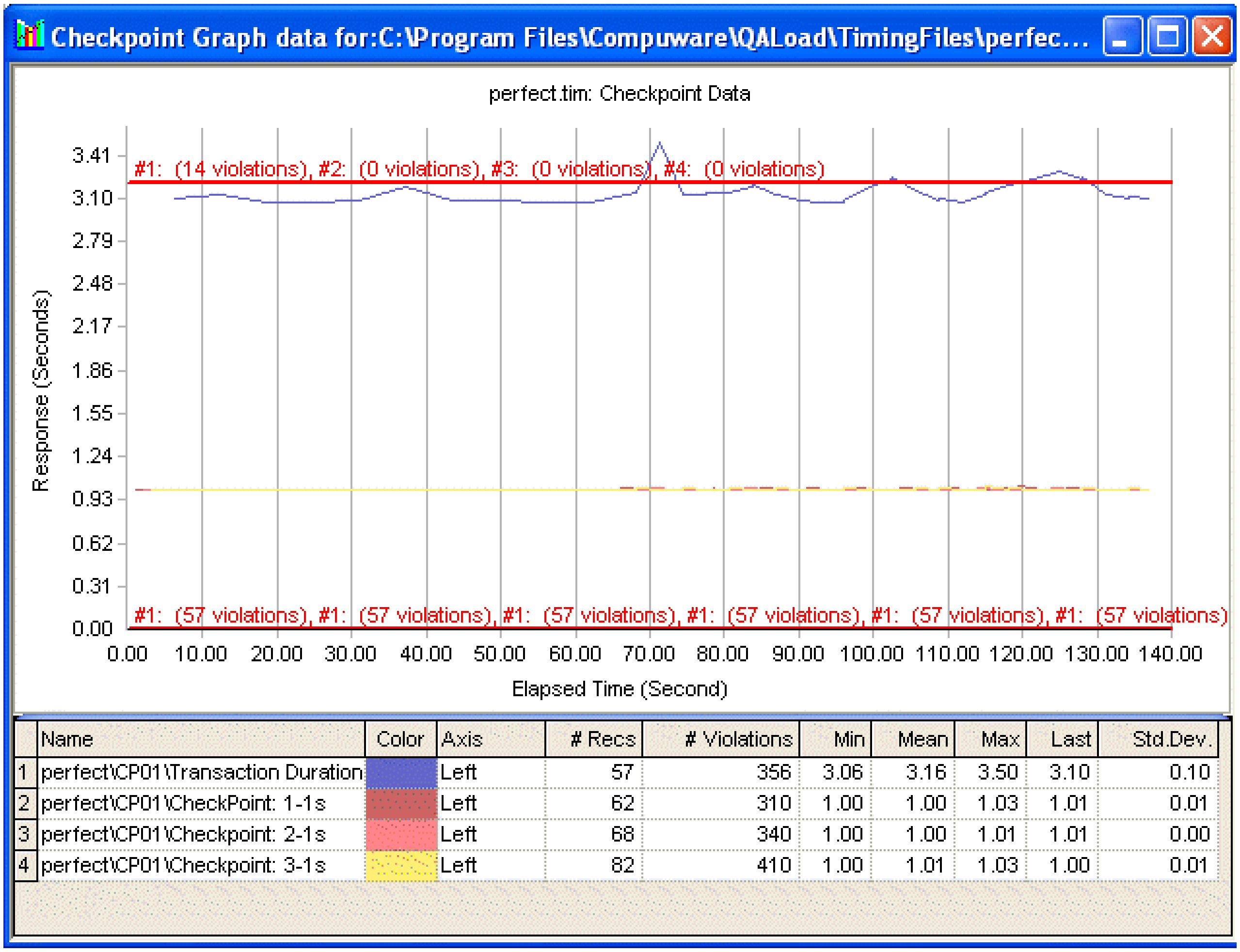
Using Thresholds

Thresholds are user-defined values that show the expected warning and critical limits for a counter or checkpoint. Thresholds help identify problem areas in the test, such as checkpoints or counters that go above or below a specified number. They actively monitor response times by indicating whether data records are surpassing user-defined expected warning and critical levels.

Thresholds are saved in the Template file. The information displays in graphs and detail reports in the detail data view.

Graphs

The selected data is displayed in a line graph format in the Data window. Graphs show thresholds as horizontal lines with the number of failed points.



	Name	Color	Axis	# Recs	# Violations	Min	Mean	Max	Last	Std.Dev.
1	perfect\CP01\Transaction Duration		Left	57	356	3.06	3.16	3.50	3.10	0.10
2	perfect\CP01\CheckPoint: 1-1s		Left	62	310	1.00	1.00	1.03	1.01	0.01
3	perfect\CP01\Checkpoint: 2-1s		Left	68	340	1.00	1.00	1.01	1.01	0.00
4	perfect\CP01\Checkpoint: 3-1s		Left	82	410	1.00	1.01	1.03	1.00	0.01

Detail Reports

Detail reports display thresholds in a Summary table and a Data table. The Summary table is a summary of raw data collected from a load test. The defined threshold limits, the percentage of failures for the threshold, and the total violations are displayed. In the Data table, the failed points are shown in red.

	Timing File	Script	Player	Checkpoint	Type	Group	#Thinned Trans	#Thinned Recs	Data Thin	Min	Mean	Max	StdDev	Median	90th Percentile	Pacing (Seconds)	VU's	Threshold	Failure Rate	Total Violations
1	perfect	CP01	ARCHIE	Transaction Duration	---	---	2048	1024	Every 2 trans	3.0640	3.1821	3.5050	0.1177	3.1390	3.2440	1	2	< 3.200000	63.77 %	653
2	perfect	CP01	ARCHIE	CheckPoint: 1-1s	---	---	2048	1024	Every 2 trans	1.0010	1.0027	1.0320	0.0037	1.0020	1.0110	---	---			
3	perfect	CP01	ARCHIE	Checkpoint: 2-1s	---	---	2048	1024	Every 2 trans	1.0010	1.0019	1.0120	0.0022	1.0010	1.0020	---	---			
4	perfect	CP01	ARCHIE	Checkpoint: 3-1s	---	---	2048	1024	Every 2 trans	1.0010	1.0047	1.0310	0.0052	1.0020	1.0120	---	---			

Checkpoint by Player Summary

	Timing File	Script	Group	Checkpoint	Type	VU	Player	#Recs	Elapsed (Seconds)	Response (Seconds)	Threshold
1	perfect	CP01	---	Transaction Duration	---	0	ARCHIE/34566	---	12.4380	3.1240	< 3.2
2	perfect	CP01	---	Transaction Duration	---	0	ARCHIE/34566	---	30.8740	3.0840	< 3.2
3	perfect	CP01	---	Transaction Duration	---	0	ARCHIE/34566	---	49.5510	3.1790	< 3.2
4	perfect	CP01	---	Transaction Duration	---	0	ARCHIE/34566	---	68.0980	3.1340	< 3.2
5	perfect	CP01	---	Transaction Duration	---	30	ARCHIE/34566	---	71.1720	3.5000	---
6	perfect	CP01	---	Transaction Duration	---	31	ARCHIE/34566	---	71.1720	3.5000	---
7	perfect	CP01	---	Transaction Duration	---	32	ARCHIE/34566	---	71.1720	3.5000	---
8	perfect	CP01	---	Transaction Duration	---	23	ARCHIE/34566	---	71.1720	3.5000	---
9	perfect	CP01	---	Transaction Duration	---	25	ARCHIE/34566	---	71.1720	3.5000	---
10	perfect	CP01	---	Transaction Duration	---	26	ARCHIE/34566	---	71.1720	3.5000	---
11	perfect	CP01	---	Transaction Duration	---	18	ARCHIE/34566	---	71.1720	3.5000	---
12	perfect	CP01	---	Transaction Duration	---	9	ARCHIE/34566	---	71.1720	3.5000	---
13	perfect	CP01	---	Transaction Duration	---	10	ARCHIE/34566	---	71.1720	3.5000	---
14	perfect	CP01	---	Transaction Duration	---	12	ARCHIE/34566	---	71.1720	3.5000	---
15	perfect	CP01	---	Transaction Duration	---	4	ARCHIE/34566	---	71.1720	3.5000	---
16	perfect	CP01		Transaction Duration		5	ARCHIE/34566		71.1720	3.5000	

Checkpoint by Player Data

Creating Thresholds

Define thresholds to show the expected warning and critical limits for a counter or checkpoint.

To create Thresholds:

1. Select **Template>Thresholds**. The **Thresholds** dialog box appears.
2. Click the appropriate group in the Workspace. Data within each group is listed in a tree-view.
3. Highlight a counter or checkpoint and drag it to the **Threshold** dialog box.

Note: You can drag entire groups or individual items to the Threshold dialog box.

4. Click Edit to name the threshold and set the threshold limits and conditions.
5. Do one of the following:
   - Click the check box in the Active column for the thresholds you want to use, then click Apply.
   - Click Activate All to use all of the thresholds, then click Apply.

Editing Thresholds

Once you create the threshold, set the threshold properties using the Edit function.

To edit Thresholds:

1. In the **Thresholds** dialog box, do one of the following:
   - Double-click the threshold you want to edit.
   - Highlight the appropriate threshold and click Edit.

   The Threshold Properties dialog box appears.

2. In the **Threshold Label** field, type a name for the threshold. This is optional.
3. In the **Limit** field, type the number for the threshold. This is used with the **Condition** you select to calculate violations.
4. In the **Limit Condition** section, choose how the threshold violations are calculated. You can select:
    - Greater than the threshold limit you defined (>)
    - Equal to the threshold limit you defined (=)
    - Less than the threshold limit you defined (<)
5. Click **OK**. The **Threshold** dialog box appears showing the limit and condition you specified.

Viewing Thresholds

Threshold information displays in graphs and detail reports in the detail data view .

Note: The data displayed and the groups available may vary, depending on the type of data that is collected during the load test.

To view thresholds in graphs:

1. In the Workspace, click the appropriate Timing File tab. Data is listed in a tree-view.
2. Click the group to view, and select the appropriate checkpoints or counters to display in the graph.

Note: You can select an entire group or individual data files.

3. Click **View> Graph** or right-click and choose **Graph** from the context menu.

The selected data displays in a line graph format in the Data window. Graphs show thresholds as horizontal lines with the number of failed points.

To view thresholds in detail reports:

1. In the Workspace, click the appropriate Timing File tab. Data is listed in a tree-view.
2. Click the group to view, and select the appropriate checkpoints or counters to display in the detail report.

Note: You can select an entire group or individual data files.

3. Click **View> Detail** or right-click and choose **Detail** from the context menu.

The detail report for the selected data displays. Detail reports display thresholds in a Summary table and a Data table. The Summary table is a summary of raw data collected from a load test. In the Data table, the failed points are shown in red.

Using the Thresholds Dialog Box

The Thresholds dialog box allows you to create system-wide thresholds for specific checkpoints and counter sets. Here you set the value that defines the limit for the threshold, which is the expected warning and critical level for the checkpoint or counter. This value is used in conjunction with the Limit Condition that you set in the Threshold Properties dialog box to determine whether data violates the threshold.

Copy: Allows you to copy a counter or checkpoint and assign a different threshold limit and condition.

Edit: Opens the Threshold Properties dialog box, where you define the threshold limit and choose how the violations are calculated.

Remove: Deletes the threshold you have highlighted.

Activate All: Selects all the thresholds so that they may be applied to the open timing file. A check is placed in the Active column.

Activate None: Removes the check in the Active column of all the thresholds so that none are applied to the open timing file.

Apply: Applies the thresholds that you activate to the open timing file.

Creating a Chart or Graph

Thinning data before graphing

Test results may contain more data than can reasonably be graphed. Thinning data before graphing provides a clearer and more manageable graph.

To set up data thinning:

1. With a timing file open, click **Tools>Options**.
2. Click the **Data Thinning** tab.
3. Type the number of data points to plot on each graph and select the method by which to graph the data points.
4. Click **OK**.

For a description of the options on this dialog box, see Options Dialog Box - Data Thinning Tab.

Graphing checkpoints

Note: A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

To graph checkpoints:

1. Open the appropriate .tim file in *QA*Load Analyze. In the Workspace, click the Checkpoints group. Checkpoint data is listed in a tree-view.
2. Select the checkpoints to graph.
3. From the **View** menu, choose **Graph**. The Select Graph dialog box appears.
4. In the Graph Type drop-down list, select from the following:
   - Line (response times versus elapsed times for the selected data.)
   - Bar (median, mean, or a percentile response time of the selected checkpoints.)

Note: The following graph types are only available when graphing a single checkpoint:

- Response Time Distribution (how the response times of a single checkpoint are distributed.)
- Cumulative Response Time Distribution (the percentage of checkpoint timings that were equal to or less than a specified value.)

Data for the selected checkpoint(s) is graphed in the Data window in the format selected in step 4.

Graphing counters

Note: A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

To graph counters:

1. Open the appropriate .tim file in *QA*Load Analyze. In the Workspace, click the **Counters** group. Counter data is listed in a tree-view.
2. Select the counter(s) to graph.
3. From the **View** menu, choose **Graph**. Data for the selected counter(s) is graphed in a line graph format in the Data window.

Graphing Player performance counters

Note: A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

To graph Player performance counters:

1. Open the appropriate .tim file in *QA*Load Analyze. Select the **Player Performance Counters** group.
2. In the Workspace, select the performance counter(s) to graph.
3. Click the **View Graph** button or right-click and choose **Graph** from the context menu. Data for the selected Agent(s) is graphed in a line graph format in the Data window.

Graphing server monitoring data

Monitoring servers is a method of load testing. QALoad provides performance counter data through three server monitoring methods:

! Remote Monitoring - Performs the monitoring of performance counters from a machine under test without the use of agent software on the machine.

- Server Analysis - Performs the monitoring of performance counters from a machine under test using the ServerVantage agent software installed on the machine.
- ServerVantage - An Availability Management application complementary to *QA*Load for service level monitoring of performance counters for applications, servers, and databases during production. ServerVantage also provides notification, event management, and reporting features.

Graphing top processes

Note: A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

To graph top processes:

1. Open the appropriate .tim file in *QA*Load Analyze. Select the **Top Processes** group.

   Note: The Top Processes group is available only if you enable the option in the QALoad Conductor Server Analysis Agent configuration screen before running a test.

2. In the Workspace, select the data point(s) to graph.
3. Click the **View Graph** button or right-click and choose **Graph** from the context menu. Data for the selected Agent(s) is graphed in a line graph format in the Data window.

Graphing Expert User Data

Note: A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. Details

To graph expert user data:

1. Open the appropriate .tim file in *QA*Load Analyze.
2. Select the **Expert User** group in the Workspace tree-view.
3. In the Workspace, select the data point(s) to graph.
4. Click the **View Graph** button or right-click and choose **Graph** from the context menu. Data for the selected Agent(s) is graphed in a line graph format in the Data window.

Customizing a Chart or Graph

Customizing a graph

Change the style and appearance of a graph using options available from the Graph toolbar. The Graph toolbar contains buttons for standard Windows operations as well as for customizing a graph's appearance.

Display the Graph toolbar by right-clicking in an open area of a graph and choosing the Toolbar option from the shortcut menu. Although it initially appears above the graph, the toolbar is completely dockable. Move the toolbar to another side of the graph, or off the graph altogether, by clicking on any unpopulated area of the toolbar and dragging it to another area.

The following features can be customized from the Graph toolbar. Click on any feature in the following list for additional information or instructions:

Graph type

Color

Grid orientation (horizontal and vertical)

Legend box

Dimension (3D or 2D)

Rotation

Z-Cluster

Font

Viewing Reports

Viewing Pre-Defined Reports

Pre-Defined Reports

QALoad Analyze provides pre-defined reports for viewing load test results without time-consuming data manipulation.

In the Workspace, select the Reports group and click the appropriate report. The reports are in HTML generated by XSL files. View them in QALoad Analyze, or directly in a Web browser.

Note: Compuware provides each of the available pre-defined reports as convenience to view the results of a load test without any data manipulation. In addition, create customized versions of these reports by selecting the appropriate group and creating detail reports and graphs.

The following reports are available. Click a report name for details.

Summary

Session

Concurrent Users

Response Time Analysis

Output

Client Throughput

Server Monitoring

Transaction Throughput

Top Ten Longest Checkpoint Durations

Worst Performing Checkpoints and Counters

Player Performance

Expert User

ApplicationVantage

Summary report

The Summary report is the primary output from each test run, one of the pre-defined reports QALoad Analyze makes available. When you open a timing file, QALoad Analyze automatically displays the Load Test Summary in the Data window. It presents timing information for each transaction in the timing file and the minimum, maximum, and median response times for each checkpoint. The output is divided into two sections. The first section presents the Summary Test Information, Test Time information, and Data Thinning and Time Range information.

The second section presents the Script Information for each script. It shows timing Summaries and Checkpoint information for each script.

Sample Summary

For a brief description of each report section, scroll down and click a section heading, for example, Test Information, in the following sample.

Summary - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages
4	1	851	76	1,104

Test Time

Start	End	Duration	Pre	User	Post
11/21/2003 - 12:48:57	11/21/2003 - 12:51:37	2 Minutes and 40 Seconds	00:00:00	00:02:40	00:00:00

Data Thinning and Time Range

Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

Script Information

CP01

Summary

	Total	Thinned
Peak Virtual Users	266	266
Transaction Pacing	1.00 Second(s)	1.00 Second(s)
Transaction Rate	43.46 Transactions Per Second	16.01 Transactions Per Second
# Errors	0	74
# Messages	6,167	1,024
Timing Data Thinning	Every 2 transactions	Every 2 transactions
Include Sleep Time	Yes	Yes

Checkpoints — Hide Thresholds

ID	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%	Threshold	Failure Rate	Total Failures
0	Transaction Duration	2,048	1,024	3.0640	3.1821	3.5050	0.1177	3.1390	3.2440	--	--	--
1	CheckPoint: 1-1s	2,048	1,024	1.0010	1.0027	1.0320	0.0037	1.0020	1.0110	--	--	--
2	Checkpoint: 2-1s	2,048	1,024	1.0010	1.0019	1.0120	0.0022	1.0010	1.0020	--	--	--

Session report

Provides summary information about the test session. The information in this report was obtained from the Conductor's configuration settings when the load test was started. To view a summary of test settings that includes changes made while the test was running, see the Summary report.

Session - mjc_aa_1_03042005_100438.TIM

Note: The following information was obtained from the Conductor's configuration settings when the load test was started. To view a summary of test settings that includes changes made while the test was running, see the Summary report.

Test Information

Summary	
Session ID Name	mjc_aa_1.ID
Conductor Build	05.05.00 Build 234
Session Duration	00:00:00
Configuration file	C:\Program Files\Compuware\QALoad\Session\Default.cfg
Total Scripts	1
Total Players	1
Total Virtual Users	10
Total Running Virtual Users	0

Script Information

mjc_aa

Summary	
Path	C:\Program Files\Compuware\QALoad\Middlewares\WWW\Scripts\mjc_aa.cpp
Middleware Type	WWW
Transactions	30
Automatic Timings	Enabled
Include Sleep Times	False
Checkpoint Thinning	Disabled
Counter Data Collection	Store in Timing File and Display in Conductor
Counter Thinning	By Script Every 1 second(s)
Sleep Factor	100%
Transaction Pacing	00:00:01.000
Service Level Threshold	00:00:00
Error Handling	Restart Transaction
Central Datapool	None
Local Datapools	
Binary Files	

Machine Information

Machines In Test

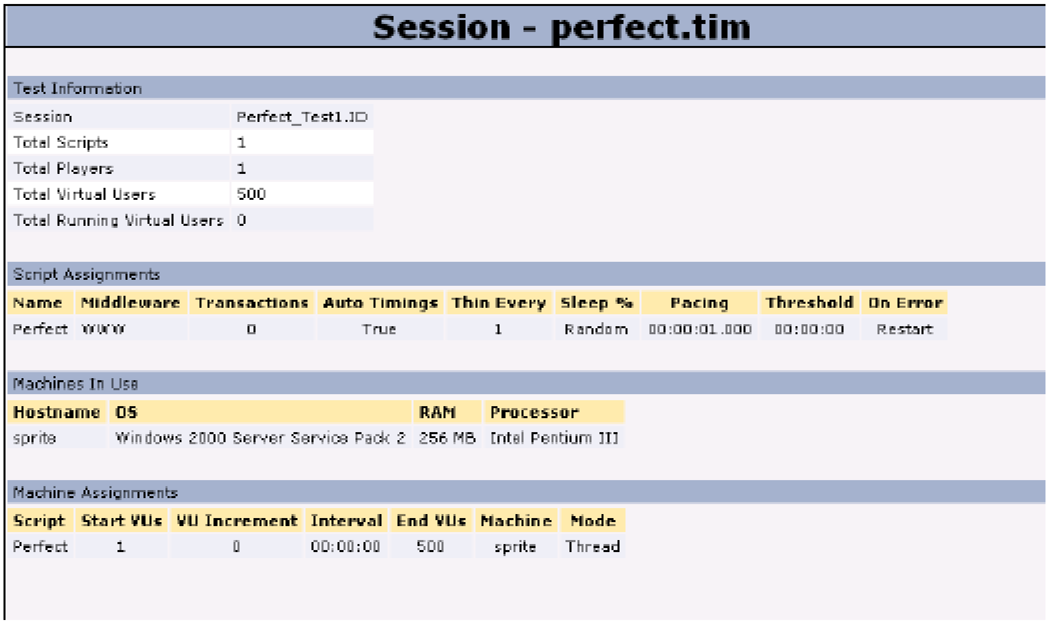
Hostname	OS	RAM	Processor
dtw114910d01	Windows XP Workstation Service Pack 1	1023 MB	Intel Pentium 4

Machine Assignments

Script	Expert User	Start VUs	VU Increment	Interval	End VUs	Machine	Mode
mjc_aa		1	1	00:00:10	10	dtw114910d01	Thread

F

or descriptions of the information provided in each section, click the sections in the following image.



Session - perfect.tim

Test Information

Session	Perfect_Test1.ID
Total Scripts	1
Total Players	1
Total Virtual Users	500
Total Running Virtual Users	0

Script Assignments

Name	Middleware	Transactions	Auto Timings	Thin Every	Sleep %	Pacing	Threshold	On Error
Perfect	WWW	0	True	1	Random	00:00:01.000	00:00:00	Restart

Machines In Use

Hostname	OS	RAM	Processor
sprite	Windows 2000 Server Service Pack 2	256 MB	Intel Pentium III

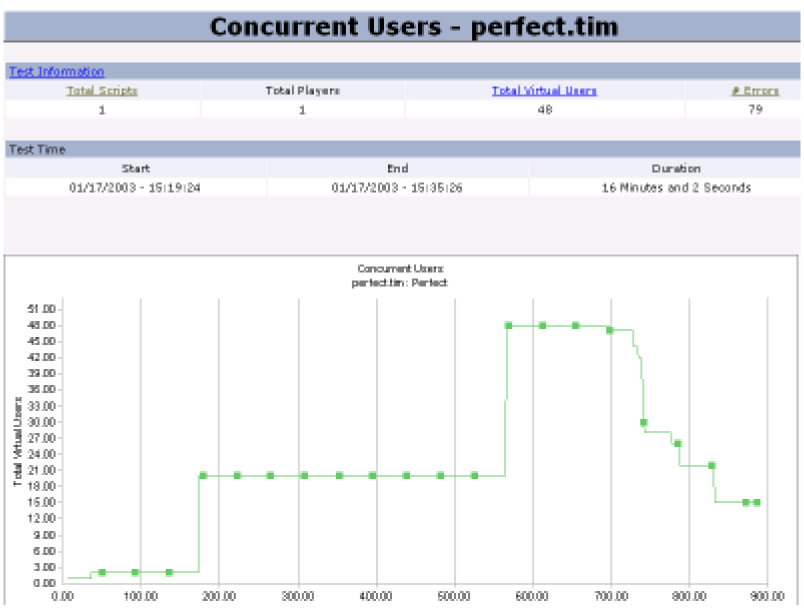
Machine Assignments

Script	Start VUs	VU Increment	Interval	End VUs	Machine	Mode
Perfect	1	0	00:00:00	500	sprite	Thread

Concurrent Users report

Displays the total number of virtual users for the test, concurrent users vs. elapsed time, as well as graphs for individual scripts that were part of the test.

Note: A totals graph will not display if the test contains only one script.



Concurrent Users - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	# Errors
1	1	48	79

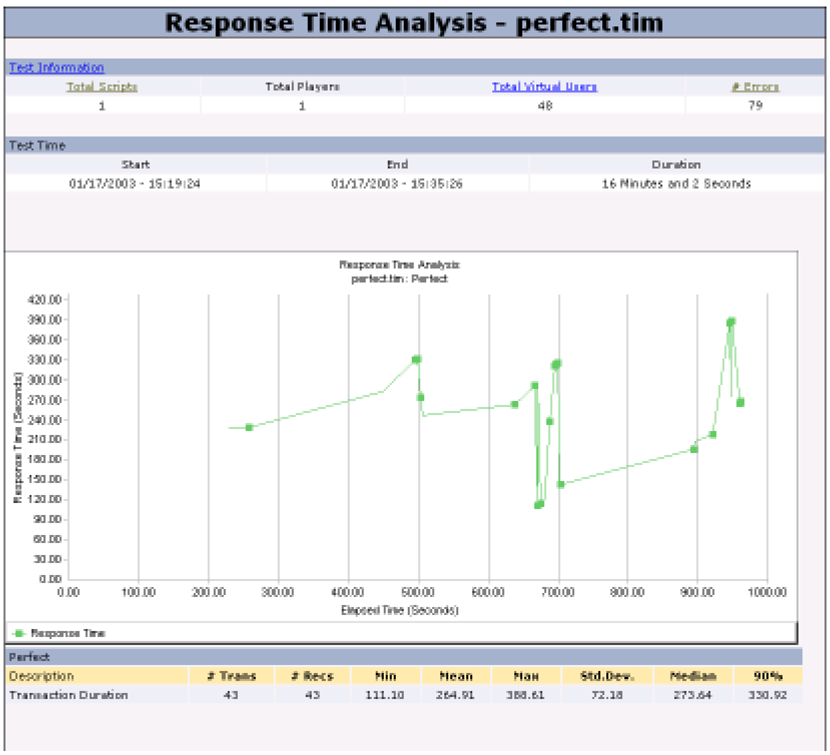
Test Time

Start	End	Duration
01/17/2003 - 15:19:24	01/17/2003 - 15:35:26	16 Minutes and 2 Seconds

Response Time Analysis report

Provides an indicator of how well a script ran. The report displays a graph of each script's transaction duration (response time vs. elapsed time) as well as the following checkpoint summary data:

#Trans: Number of data points used to calculate the statistics.
#Recs: Number of data records. This value, if different from the value of #Trans, reflects the number of checkpoint records that are used for analysis after data thinning has been applied.
Min: Minimum recorded response time.
Max: Maximum recorded response time.
Std. Dev: Standard deviation of all response times. A large standard deviation indicates a wide variance in response times.
Median: Median response time, in seconds.
nth%: n percent of the responses have a value less than the value shown.



Response Time Analysis - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	# Errors
1	1	48	79

Test Time

Start	End	Duration
01/17/2003 - 15:19:24	01/17/2003 - 15:35:26	16 Minutes and 2 Seconds

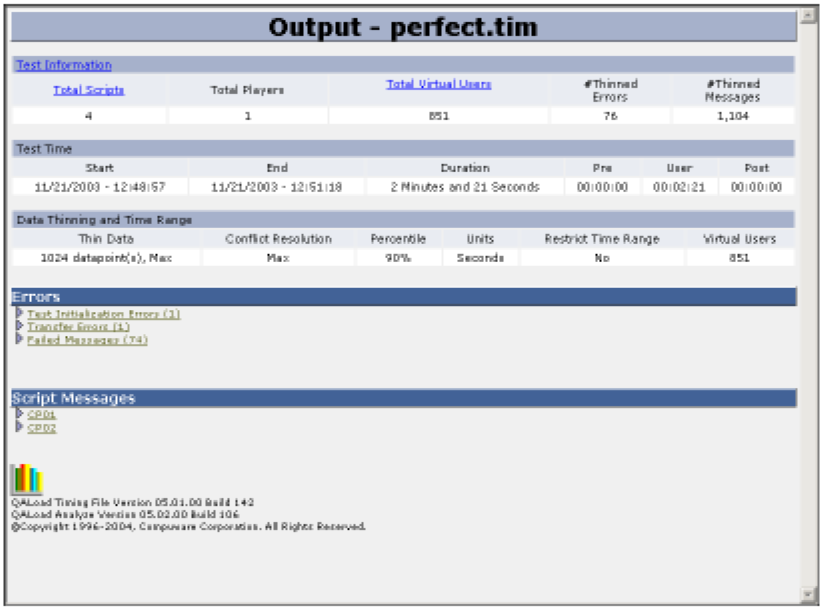
Perfect

Description	# Trans	# Recs	Min	Mean	Max	Std.Dev.	Median	90%
Transaction Duration	43	43	111.10	264.91	388.61	72.18	273.64	330.92

Output report

Provides a cumulative list of all errors, sorted by script and occurrence in time, that occurred during the course of a load test.

> Note: Failed messages are included in the errors count that appears in the Test Information section of the report, but are detailed in the Script Messages section.

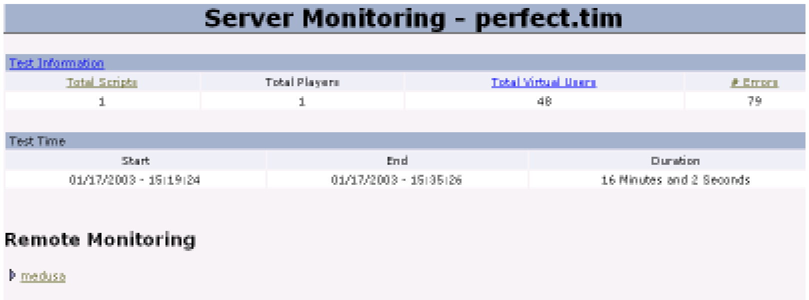


Client Throughput report

Provides a graph of HTTP Reply analysis for key HTTP counters, HTTP counter vs. elapsed time.

Server Monitoring report

Server monitoring is a component of load testing. QALoad provides performance counter data through three server monitoring methods: Remote Monitoring, ServerVantage, and Server Analysis Agent monitoring.



Server Monitoring - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	# Errors
1	1	48	79

Test Time

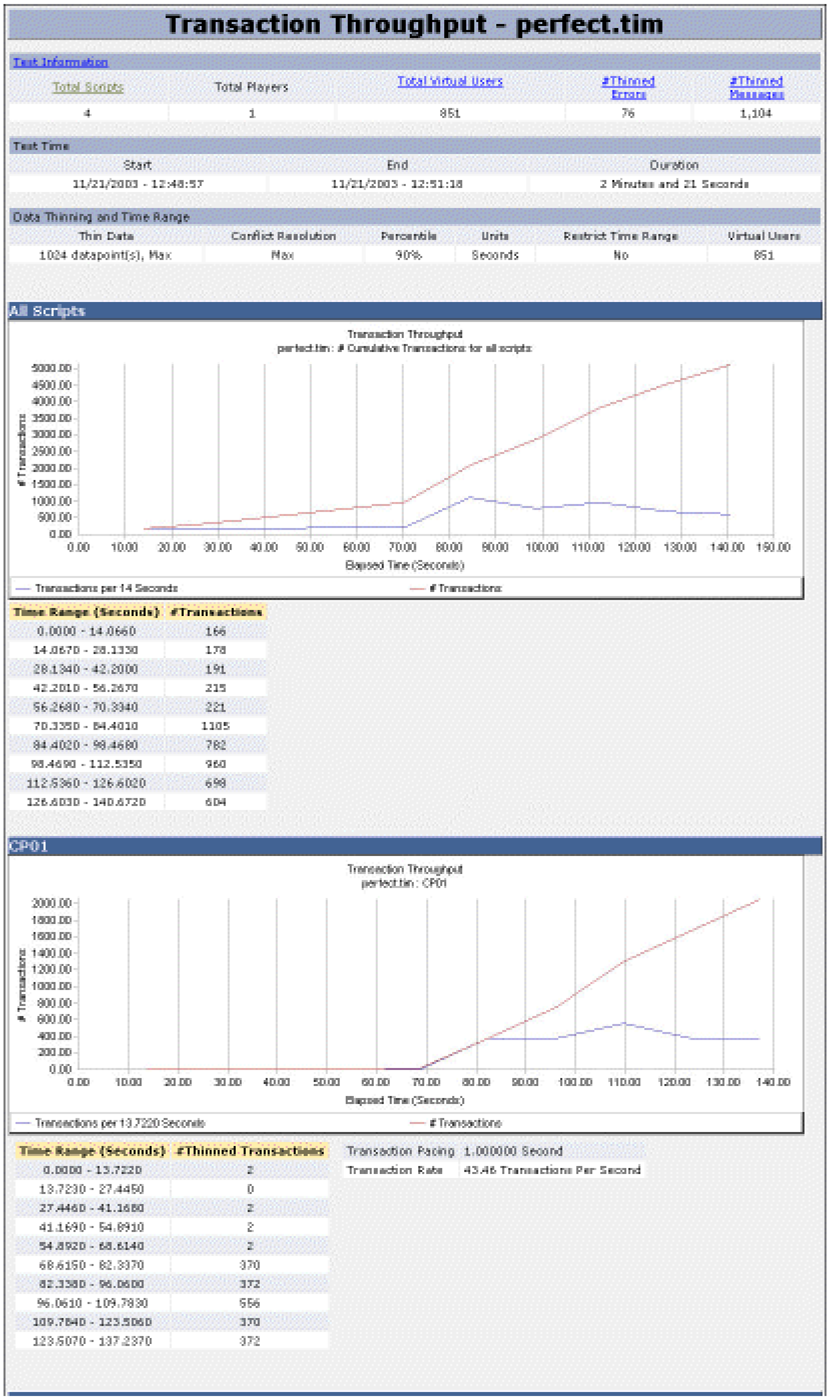
Start	End	Duration
01/17/2003 - 15:19:24	01/17/2003 - 15:35:26	16 Minutes and 2 Seconds

Remote Monitoring

- medusa

Transaction Throughput report

Provides the cumulative number of transactions over elapsed time for each script and for the total test.



Transaction Throughput - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages
4	1	851	76	1,104

Test Time

Start	End	Duration
11/21/2003 - 12:48:57	11/21/2003 - 12:51:18	2 Minutes and 21 Seconds

Data Thinning and Time Range

Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

All Scripts

Time Range (Seconds)	#Transactions
0.0000 - 14.0660	166
14.0670 - 28.1330	178
28.1340 - 42.2000	191
42.2010 - 56.2670	215
56.2680 - 70.3340	221
70.3350 - 84.4010	1105
84.4020 - 98.4680	782
98.4690 - 112.5350	960
112.5360 - 126.6020	698
126.6030 - 140.6720	604

CP01

Time Range (Seconds)	#Thinned Transactions
0.0000 - 13.7220	2
13.7230 - 27.4450	0
27.4460 - 41.1680	2
41.1690 - 54.8910	2
54.8920 - 68.6140	2
68.6150 - 82.3370	370
82.3380 - 96.0600	372
96.0610 - 109.7830	556
109.7840 - 123.5060	370
123.5070 - 137.2370	372

Transaction Pacing	1.000000 Second
Transaction Rate	43.46 Transactions Per Second

Top Ten Longest Checkpoint Durations report

Provides graphs and lists details about checkpoints that had the longest checkpoint duration during the test. Checkpoints with longest durations are those that consumed the most amount of time during the test. This report contains the following sections:

- A summary section with overview information about the test.
- A bar graph of the ten longest checkpoint durations in the test, followed by details for each checkpoint in the graph. These checkpoints can originate in any script that was included in the test.
- Bar graphs for each script that show up to the ten longest checkpoint durations, followed by details for each checkpoint in the script.

The report is generated by Analyze only if each script has at least one checkpoint other than the duration checkpoint. The data provided in the report can be used as a starting point to identify performance problems.

Note: Transaction duration checkpoints are not included in the report.

Top Ten Longest Checkpoint Durations - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages
4	1	851	76	1,104

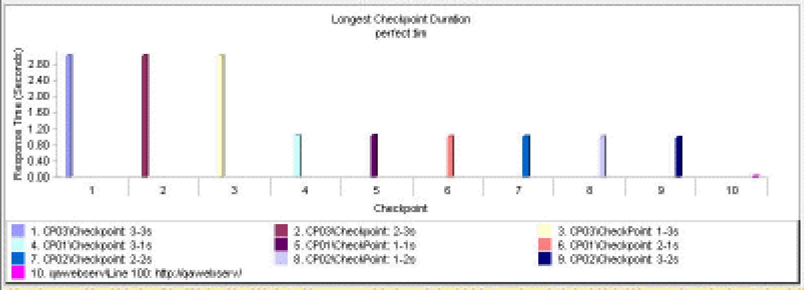
Test Time

Start	End	Duration
11/21/2003 - 12:48:57	11/21/2003 - 12:51:18	2 Minutes and 21 Seconds

Data Thinning and Time Range

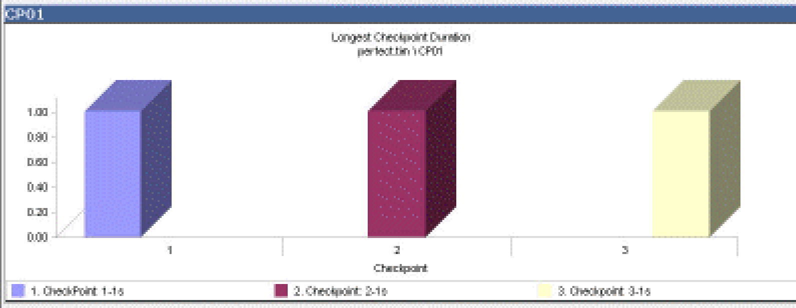
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

Longest Checkpoint Duration Summary



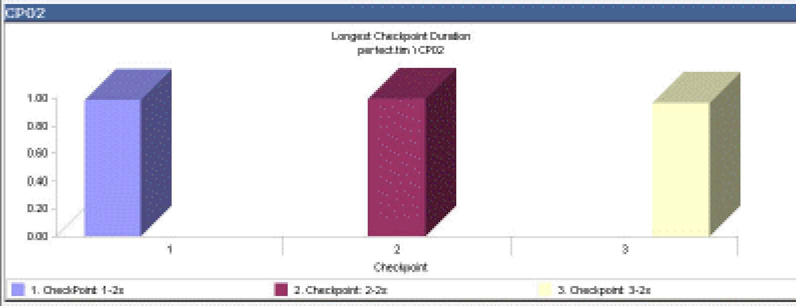
	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1	CP03\Checkpoint: 3-3s	1,024	75	3.0040	3.0054	3.0240	0.0035	3.0040	3.0050
2	CP03\Checkpoint: 2-3s	1,024	74	3.0040	3.0048	3.0240	0.0026	3.0040	3.0050
3	CP03\CheckPoint: 1-3s	1,024	82	3.0040	3.0046	3.0150	0.0020	3.0040	3.0050
4	CP01\Checkpoint: 3-1s	2,048	82	1.0010	1.0062	1.0310	0.0070	1.0020	1.0120
5	CP01\CheckPoint: 1-1s	2,048	62	1.0010	1.0046	1.0320	0.0060	1.0020	1.0120
6	CP01\Checkpoint: 2-1s	2,048	68	1.0010	1.0032	1.0120	0.0039	1.0020	1.0110
7	CP02\Checkpoint: 2-2s	1,024	819	0.0100	0.9924	2.0030	0.5679	1.0020	1.7650
8	CP02\CheckPoint: 1-2s	1,024	839	0.0000	0.9883	2.0030	0.5760	0.9810	1.7830
9	CP02\Checkpoint: 3-2s	1,024	832	0.0100	0.9662	2.0030	0.5956	0.9220	1.8130
10	qawebserv\Line 100: http://qawebserv/	1,024	1,024	0.0000	0.0027	0.0300	0.0045	0.0000	0.0100

CP01



	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1	CheckPoint: 1-1s	2,048	62	1.0010	1.0046	1.0320	0.0060	1.0020	1.0120
2	Checkpoint: 2-1s	2,048	68	1.0010	1.0032	1.0120	0.0039	1.0020	1.0110
3	Checkpoint: 3-1s	2,048	82	1.0010	1.0062	1.0310	0.0070	1.0020	1.0120

CP02

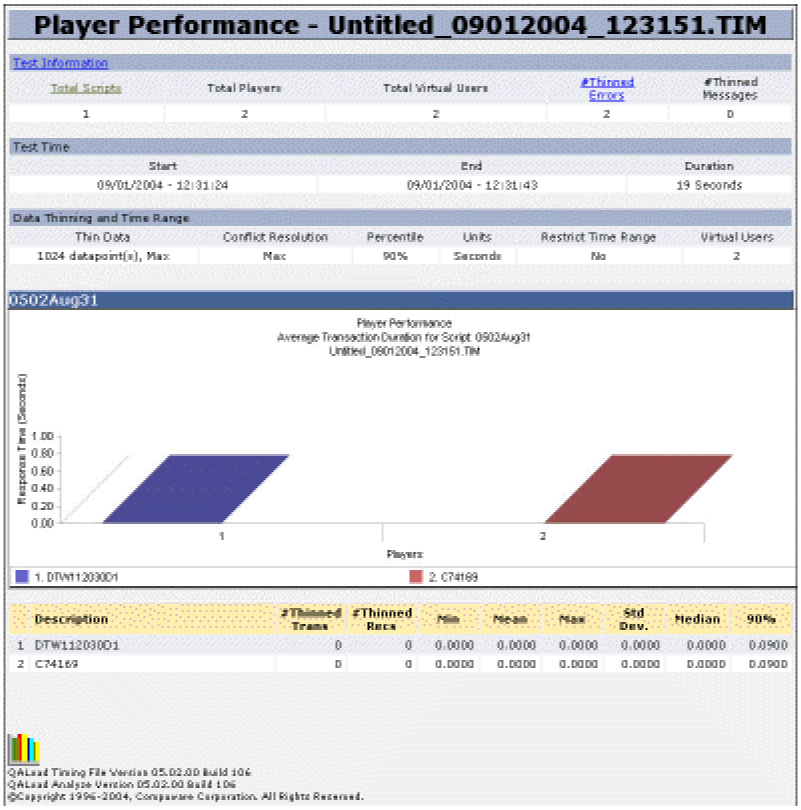


	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1	CheckPoint: 1-2s	1,024	839	0.0000	0.9883	2.0030	0.5760	0.9810	1.7830
2	Checkpoint: 2-2s	1,024	819	0.0100	0.9924	2.0030	0.5679	1.0020	1.7650
3	Checkpoint: 3-2s	1,024	832	0.0100	0.9662	2.0030	0.5956	0.9220	1.8130

CP03

Player Performance report

Displays transaction durations in a graph format by player machine. This report helps identify individual player machines that have poor test results. In addition to the bar graph that plots the average transaction duration for each player machine, the report also includes summary data for the overall test, and details for each player machine. This report is generated by Analyze only if two or more player machines were used in the test.



Worst Performing Checkpoints and Counters report

This report provides graphs and lists details about checkpoints and counters that had the worst performance during the test. Performance is based on the thresholds you define.

Checkpoints with the worst performance are calculated using the average response time for each checkpoint. Counters with the worst performance are those that consume the most amount of time during the test. Checkpoints with the most errors and counters with the highest failure rates are listed first. The data provided is a starting point for identifying performance problems.

> Note: The report is generated by Analyze only if a threshold is defined and if the threshold is violated by the data. Thresholds that are not violated do not appear in the report.

The bar graphs for each script show up to ten of the longest checkpoint durations and are followed by details for each checkpoint in the graph. The checkpoints can originate in any script that was included in the test. The report contains the following sections:

- A summary section with overview information about the test.
- Bar graphs for checkpoints and bar graphs for counters showing the following:

- Summary by percentage of violations - Shows the number of points that exceeded the threshold divided by the total number of data points. For example, in the report for counters below, the total failure of 8 divided by the total records of 60, yields a failure rate of 13.3 percent.
- Summary by severity - Shows the percent of time during the test that the data is in violation of the threshold. This calculation uses a weighted average to determine the percentage. For example, in the report for counters below, when the weighted average is used to calculate the severity, the failure rate is 7.6 percent and 6.5 percent.

Since the first method measures the failure rate by percentage of violations, and the second method measures failures by the amount of time the data is in violation, the numbers can differ greatly between the two methods. It is possible for a data set to have a 5 percent failure rate when calculated by percentage of failures, and an 80 percent failure rate when calculated by severity. This can indicate that an error deviating significantly from the norm may be a more notable failure than a greater number of failures.

In the example below, the counter summary by percent of violations for Server Analysis shows a total of 8 failures out of 60 records, for a failure rate of 13.3 percent. The percentage of violations for Remote Monitoring has a total of 3 failures in 41 records, for a failure rate of 4.2 percent.

When measured by severity, however, the failure rate for Remote Monitoring is more serious than for Server Analysis. Here, the failure rate for Remote Monitoring is 7.6 percent compared to the rate 6.5 percent for Server Analysis.

Note: Transaction duration checkpoints are not included in the report.

Worst Performing Checkpoints and Counters - perfect.tim

Test Information

Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages
4	1	851	76	1,104

Test Time

Start	End	Duration
11/21/2003 - 12:48:57	11/21/2003 - 12:51:37	2 Minutes and 40 Seconds

Data Thinning and Time Range

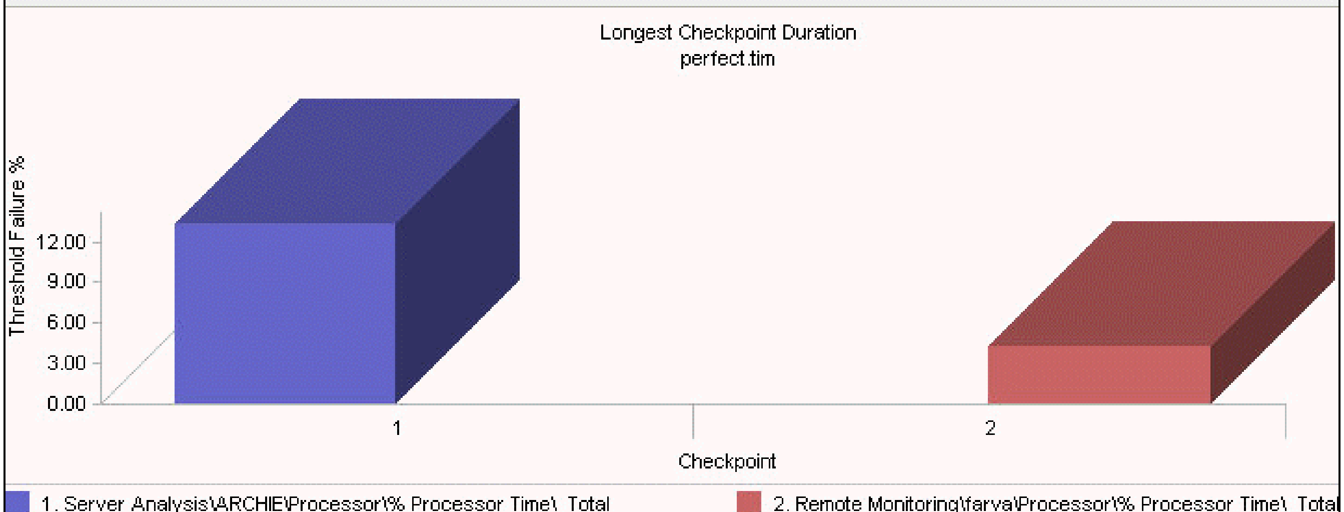
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

Checkpoints

Worst Performing Checkpoints have not been found. There are no checkpoints that violate their respective thresholds.

Counters

Worst Performing Counter Summary (by % violations)



	Description	#Thinned Trans	#Thinned Recs	Max	Threshold	Failure Rate	Total Failures
1	Server Analysis\ARCHIE\Processor\% Processor Time_Total	0	60	57.7889	> 18.0	13.3%	8
2	Remote Monitoring\farva\Processor\% Processor Time_Total	0	71	100.0000	> 34.0	4.2%	3

Worst Performing Counter Summary (by severity)

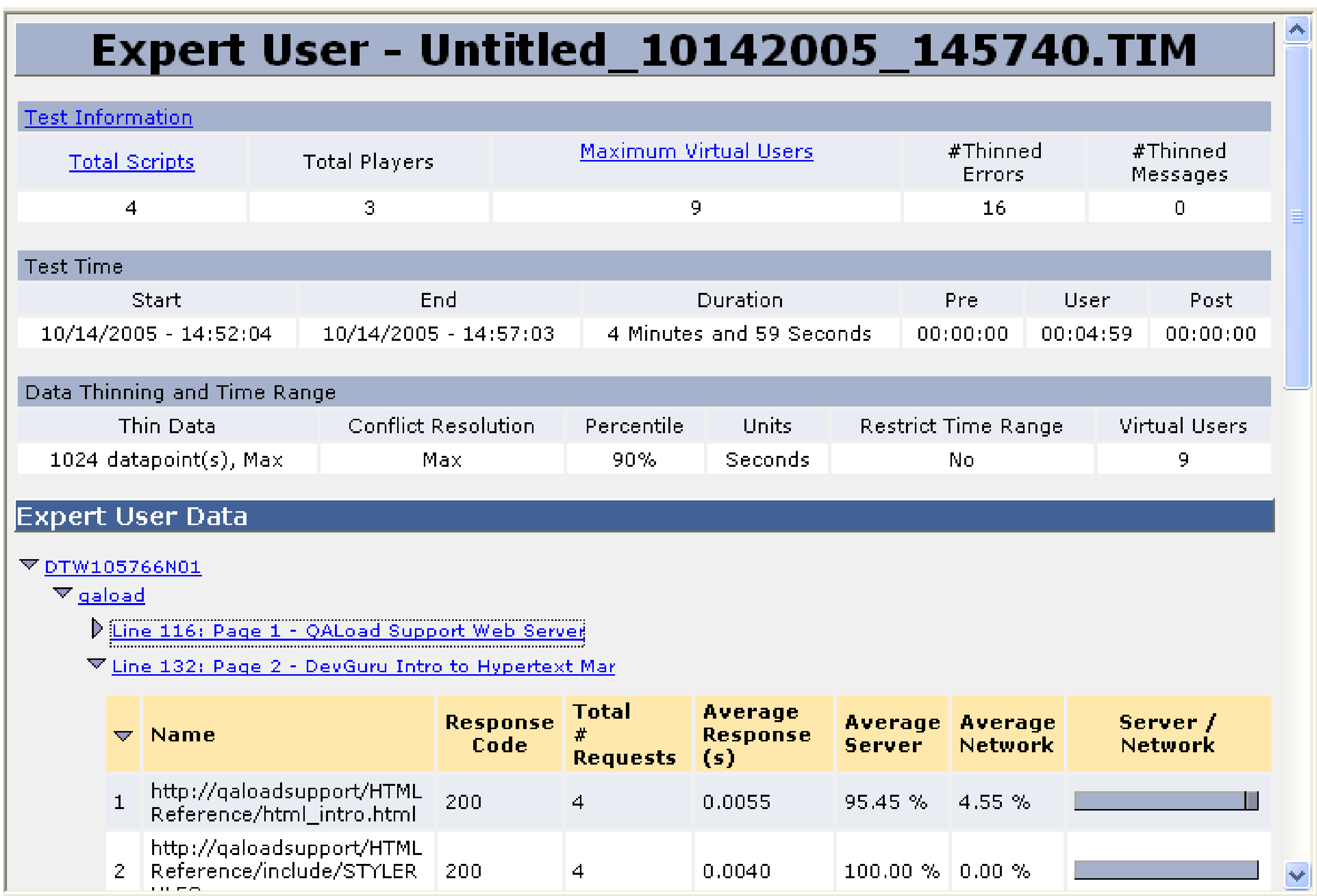
Expert User Report

Displays information on the scripts that ran on each player machine with the Expert User option enabled. It provides a timing breakdown of the individual components, such as HTML, images, and objects, of the web pages that were requested during a test. The Expert User report shows how much time it took to download a particular component of a web page from the server. It also shows the percentage of network and server time for the request.

The Expert User report contains a summary section and a detail section. The Summary section at the top of the report displays overview information about the test for each QALoad Player instance.

The detail section displays the main requests and each subrequest made when the script executes. Main requests are made when Navigate_to(), Click_On(), Post_to(), DO_http(), or DO_https() are executed in a WWW script. The subrequests, or Web components, that make up the main page can include html, css, js pages, and so forth.

The percentage of server and network time displays in the Average Server and Average Network fields, with a graphic representation in the Server/Network field. This information can help you determine whether web pages with a high response time are having server- or network-related performance problems. If an exceptional amount of time is being spent on the server, you can monitor the server that is under test using a server monitoring tool such as ServerVantage or QALoad's Remote Monitoring options. If too much time is being spent on the network, you can monitor the network under test using a tool such as ApplicationVantage.



ApplicationVantage Report

Displays information on the scripts that ran on each player machine. It contains a summary section with overview information about the test, a graph of the transaction response time of the script.

> Note: If Expert User data was collected (WWW only), the Network and Server percentage data is also displayed.

A detail section displays the beginning and ending time for the trace files produced by the scripts running on the ApplicationVantage player machines. If more than one ApplicationVantage Player machine ran during the test period, a separate section appears for each one.

Application Vantage File Report - Untitled_10142005_145740.TIM

Test Information

Total Scripts	Total Players	Maximum Virtual Users	#Thinned Errors	#Thinned Messages
4	3	9	16	0

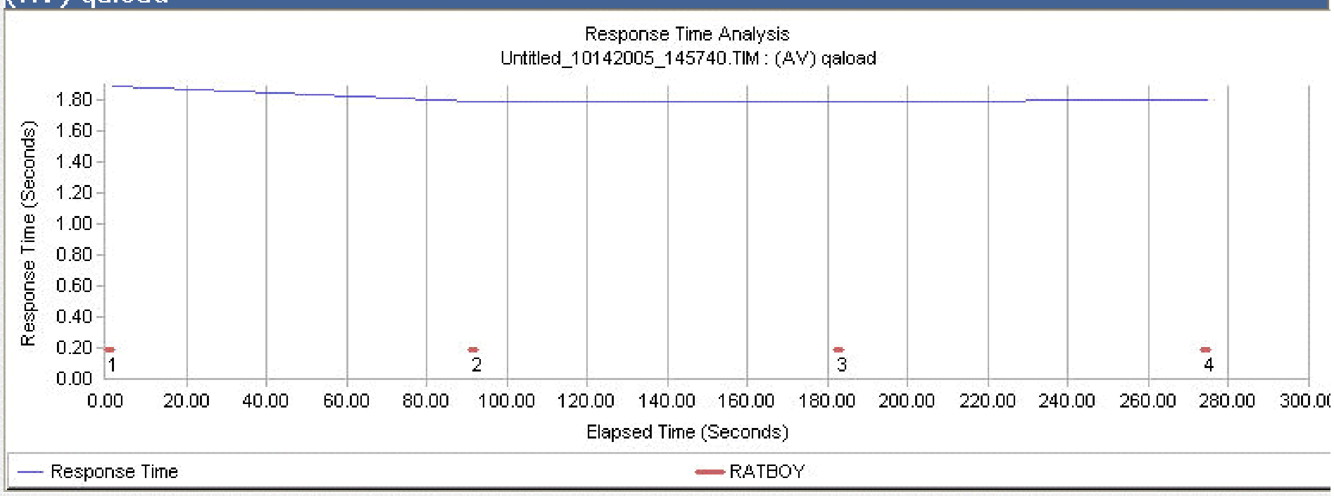
Test Time

Start	End	Duration
10/14/2005 - 14:52:04	10/14/2005 - 14:57:03	4 Minutes and 59 Seconds

Data Thinning and Time Range

Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	9

(AV) qaload



RATBOY

#	File Name	Begin Time	End Time	Total Time
1	qaload_AV_20051410_145406_1.opx	0.01	1.90	1.89
2	qaload_AV_20051410_145537_2.opx	90.55	92.34	1.79

Publishing or Sharing Test Results

Exporting test data

Convert test data into three convenient formats for viewing or exporting:

HTML — Export data in a detail view or graph to HTML files for convenient viewing in a default Web browser or for sending as attachments in an email message. See Exporting data to HTML for instructions.

Comma-separated value (CSV) — Export data in a detail view to comma-separated value (CSV) files which can be imported into popular spreadsheet applications. See Exporting data to CSV for instructions.

RIP — Any time a user fails during load testing, QALoad Analyze generates a RIP file containing user errors. If a timing file has RIP file data, you can export the RIP file to the working folder and view it in QALoad Analyze or the QALoad Script Development Workbench. See Exporting RIP file data for instructions.

ApplicationVantage (AV) trace files - When a timing file has ApplicationVantage data, you can export the ApplicationVantage trace files to a working directory and view them within ApplicationVantage. See Exporting ApplicationVantage Trace Files for instructions.

Exporting data to HTML

To export data from a detail view to HTML:

1. Open a timing file.
2. Generate a detail view or graph.
3. Click anywhere in the detail view or graph, making it active.
4. From the **File** menu, choose **Export>Data**. The Save As dialog box appears.
5. Navigate to the appropriate location for saving the HTML file and name the file.
6. Select **Web Page (*.htm;*.html)** as the file type and click **Save**.

Exporting data to CSV

To export data from the Detail view to a CSV file (*.csv):

1. Open a timing file.
2. Generate a detail view.
3. Click anywhere in the detail view making it active.
4. From the **File** menu, choose **Export>Data**. The Save As dialog box appears.
5. Navigate to the appropriate location for saving the file. In the **File name** field, type a name for the file.
6. Select **CSV (comma delimited) (*.csv)** as the file type and click **Save**.

Exporting RIP file data

Note: If a timing file does not contain any RIP data, then a RIP Files group will not exist in the Workspace.

To export the RIP file data to the working folder:

1. Open a timing file.
2. In the Workspace, click the RIP Files group.
3. In the tree view, select the appropriate RIP files check box.
4. Right-click on the selected files and choose **Export**. The Browse For Folder dialog box appears.
5. Select the folder you wish to export the RIP file data to. The default is the working folder.
6. Click **OK**. Analyze exports the RIP file to the working folder.

Exporting ApplicationVantage Trace Files

You can export ApplicationVantage trace files to a working folder or to ApplicationVantage.

Note: If a timing file does not contain any ApplicationVantage data, then the ApplicationVantage group does not exist in the Workspace.

To export the ApplicationVantage data to a working folder:

1. Open a timing file.
2. In the Workspace, click the ApplicationVantage group.
3. In the tree view, select the appropriate ApplicationVantage files check box.
4. Right-click on the selected files and choose **Export to File**. The Browse For Folder dialog box appears.
5. Select the folder you wish to export the ApplicationVantage file data to. The default is the working folder.
6. Click **OK**. Analyze exports the ApplicationVantage file to the working folder.

To export the ApplicationVantage data to ApplicationVantage:

1. Open a timing file.
2. In the Workspace, click the ApplicationVantage group.
3. In the tree view, select the appropriate ApplicationVantage files check boxes.
4. Do one of the following:
   - Right-click on the selected files and choose Export to ApplicationVantage.
   - Double-click the file to export.

   The trace files are loaded into the ApplicationVantage database and then opened in ApplicationVantage.

Sending email messages with test data

If you are using a Microsoft mail program, QALoad Analyze can send an email message with a timing file or pre-defined report attached. The recipient(s) of the message will be able to open the files in a Web browser.

To email pre-defined reports:

1. Choose **File>Send**.
2. In the **Send** dialog box, select reports, views, and timing files from their respective tabs and click **Add** to add them to the list of items you want to send.
3. In the **Send To** field, choose **Email Recipient**.
4. (optional) Click the **Zip to file** check box to send the files in the compressed .zip format. Type a name for the .zip file in the adjacent field.
5. Click **OK**. Analyze creates a new Outlook email message that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file as attachments, or a single .zip file that contains those files as an attachment. Address the email, add message text, and send the message.

Creating a .zip file of test results

You can create a .zip file to conveniently package all test data into one file for sending to others or storing locally. Analyze creates a file in .zip format, which you can either save to a location on your computer or send as an attachment to an email.

To create a .zip file:

1. Choose **File>Send**.
2. In the **Send** dialog box, select reports, views, and timing files from their respective tabs and click **Add** to add them to the list of items you want to include in the .zip file.
3. In the **Send To** field, choose **Email Recipient** to email the zip file or choose **File** to save the file on your computer.
4. Click the **Zip to file** check box to send the files in the compressed .zip format. Type a name for the .zip file in the adjacent field.
5. If you chose **File** in step 3, type the path of the location for the .zip file or click the browse button [**...**] to select a location.
6. Click **OK**. Depending on which option you chose in step 3, Analyze performs one of the following actions:
    - If you chose Email Recipient, Analyze creates a new Outlook email message that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file as a single, compressed .zip file attachment. Address the email, add message text, and send the message.
    - If you chose File, Analyze creates a single, compressed .zip file in the location you specified in step 5 that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file.

Viewing reports

View reports generated by QALoad Analyze on a machine with QALoad installed or on any machine with a Web browser. In order to save the contents of a timing file's working folder, when viewing reports, clear the Remove XML Working Folder option. To properly set this option, see the Workspace tab on the Options dialog box. For more information, see Options Dialog Box - Workspace Tab.

Viewing reports on a machine with QALoad Analyze

To view reports in QALoad Analyze, click the Summary report button or any of the pre-defined report buttons in the QALoad Analyze Workspace. See Load Test Summary for a quick introduction to viewing reports.

Viewing reports on a machine without QALoad Analyze

To view reports in a Web browser, copy the entire working folder for the timing file onto the machine. The following files are required (where <Summary> represents the name of the report):

- <*Summary*>.htm
- <*Summary*>.xml
- <*Summary*>.xsl

In addition, the Microsoft XML version 4.0 parser (provided with QALoad) is required to view QALoad reports. View any of the pre-defined reports by clicking the <Summary>.htm file to launch a report with the assistance of the associated XML and XSL support files.

Other ways to view test data

View not only pre-defined reports, but also timing file detail views and graphs by exporting or sending email messages with test data to another machine. Click the following links for more information:

- Exporting Test Data
- Sending Email Messages with Test Data

Viewing test results in a Web browser

An important part of the load testing process is viewing and studying the results of a test. You can view the results of a load test not only on a machine where QALoad is installed, but also on any machine with a Web browser. QALoad Analyze provides pre-defined reports as well as .xml and .xsl files which can be customized to meet desired specifications.

When you open a timing file, QALoad Analyze generates a working folder containing all supporting files, reports, and images generated from that timing file. This folder is located in the directory `\Program Files\Compuware\QALoad\TimingFiles\xxx.xml.source` where <xxx> is the name of the timing file.

The following files are found in the working folder:

File Name	Description
`<timingfile>.xml.source`	Working folder generated in the Reports folder when opening a timing file. The working folder name is always the <name of the timing file> with a .xml.source extension.
`<timingfile>.xml`	Original timing file with just enough information to create the QALoad Analyze pre-defined reports. It is a representation of the timing file, <timingfile>.tim.
`<timingfile>.complete.xml`	Original timing file containing all data collected during a load test. It can be an extremely large file. Use this file if creating a report using XSL that required this data.
`summary.htm`	Use this HTM file to view the Summary report (or any other available pre-defined report) in any Web browser.
`summary.xml`	Generated XML file for the Summary report (or any other

	available pre-defined report.)
`summary.xsl`	Generated XSL file for the Summary report (or any other available pre-defined report.) Translates the .xml file specifying HTML as its output and generates the HTML report. Use this file to customize the reports by writing in .xsl.
`default.htm`	Report which provides a main screen to launch any other pre-defined reports. Uses nav.htm for the navigation frame.

When closing a timing file, either keep all of the reports generated from the timing file in the working folder, or delete them. To set this option, see the Workspace tab on the Options dialog box.

To view load test results in a Web browser, click: How to View Reports.

Index