QALoad 05.06

Using the Player, Script Development Workbench, and Analyze

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:

Access is limited to authorized users. Use of this product is subject to the terms and conditions of the user's License Agreement with Compuware Corporation.

U.S. GOVERNMENT RIGHTS

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in Compuware Corporation license agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii)(OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable. Compuware Corporation.

This product contains confidential information and trade secrets of Compuware Corporation. Use, disclosure, or reproduction is prohibited without the prior express written permission of Compuware Corporation.

Compuware, ActiveAnalysis, ActiveData, Interval, QACenter, QADirector, QALoad, QARun, Reconcile, TestPartner, TrackRecord, and WebCheck are trademarks or registered trademarks of Compuware Corporation.

Acrobat® Reader copyright © 1987-2002 Adobe Systems Incorporated. All rights reserved. Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.

All other company or product names are trademarks of their respective owners.

US Patent Nos.: Not Applicable.

Doc. CWQLHX560
November 7, 2007

Table Of Contents

Player

Overview of the QALoad Player

The QALoad Player simulates one or more virtual-users running C++ or Java based scripts, depending upon which middleware environment was used for the test. These scripts mimic user activities to load test the application, network, and server components of a client-server system.

The QALoad Player simulates 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 functions 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. When the Conductor process closes for any reason during a load test, the associated Player processes terminate.

Citrix and SAP 6.20/6.40 scripts play back in a virtual user window on the desktop. For SAP, it is possible to 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.

Transfer 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 QALoad 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.

Validating Scripts in the Player

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.

Note: During validation of SAP scripts, do not minimize the SAP window. If the window is minimized, the validation may fail. This problem does not occur if you select the Hide Graphical User Interface for SAP Users option by clicking Browse [...] in the Type column of the Script Assignment tab in the Conductor. This SAPGUI option runs SAP on an alternate desktop that is not visible.

To configure the Player for validation:

1. In the Script Development Workbench, click Options>Workbench and select the Script Validation tab.
2. Select the Automatically Recompile check box if you want QALoad to compile a script before attempting to validate it. QALoad 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, if applicable.
5. Type a value in the Wait up to field. This is the number of seconds that the QALoad 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 QALoad 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 a script in the Player:

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.

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.

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 to begin script validation. Player messages will display below.

Abort: Click to stop all virtual users immediately.

Exit: Click 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.

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++ 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.

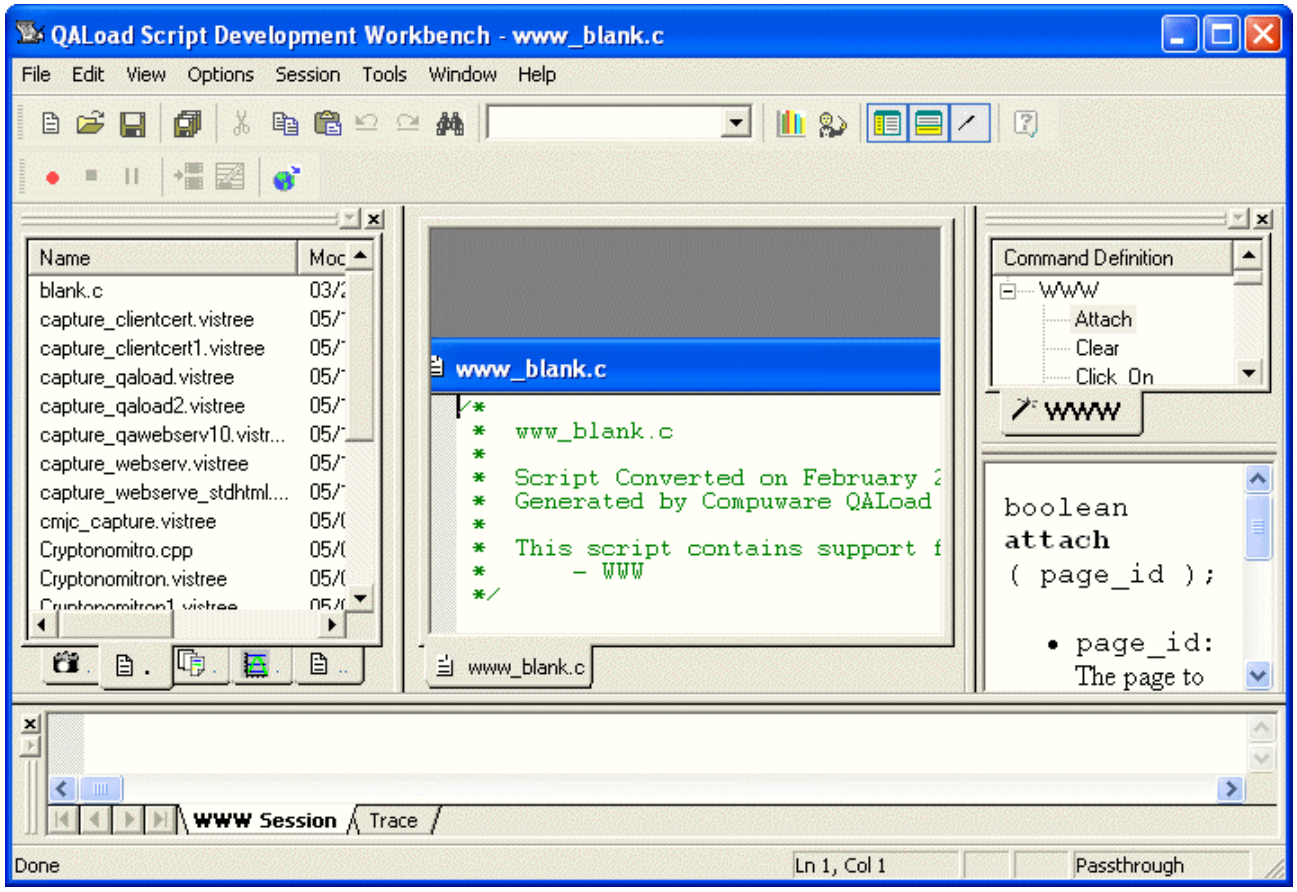
In short, to produce a usable test script you:

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.

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

Accessing the QALoad Script Development Workbench

To access the Script Development Workbench:

1. From the Windows taskbar, click the Start button.
2. Choose Programs>Compuware>QALoad>Script Development Workbench.

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 Automatically Convert Capture on the Workbench Configuration tab and Automatically Compile Scripts on the Compiler Settings tab.
5. Click OK to save your settings.

Using EasyScript 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 cannot 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 changes.

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

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 QALoad 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 QALoad Client Certificate. QALoad 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 QALoad 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.

Using Middleware Sessions

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 application(s) to record.

Opening a Middleware Session

To access the Script Development Workbench and open a middleware session:

1. Click Start>Programs>Compuware>QALoad\ Script Development Workbench.
2. Choose the middleware name from the Session menu or by click the appropriate button on the toolbar. The Default Session Prompt opens.

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.

3. Click OK.

Setting Conversion Options

Before you begin recording, you can set options to automatically customize your script during conversion. Compuware recommends you set conversion options before recording, then use the option to automatically convert your capture files to scripts. Details

1. Access the Script Development Workbench. Details
2. From the Session menu, select the appropriate middleware or start a Universal session.
3. Select Options>Convert to open the appropriate Convert Options dialog box.
4. In the Session Options tree, click Shared Convert Options and set any applicable options. This tree-view contains common options that apply to all the middleware environments QALoad supports.
5. Click the middleware-specific convert options in the Session Options tree, and set any appropriate middleware-specific options.

Notes:

- For the Citrix Web Interface environment, you must select options in both the Citrix Convert Options and the WWW Convert Options in the tree view.

- For WWW, refer to Using the WWW Convert Options.

6. When you are finished, click OK to save the current settings.

Hint: Press F1 from any middleware options tab for a description of available options.

ADO

Recording ADO Sessions

Click the ADO button on the toolbar to open an ADO session.

To set ADO recording options and begin recording:

1. Click Options>Record on the menu bar. The Session Options dialog box appears.
2. Select the appropriate options in the right-hand pane, then click OK.
3. Click the Record button on the Session toolbar.

Setting ADO Convert Options

To set conversion options for an ADO session:

1. Choose Options>Convert in the Script Development Workbench menu bar. The Session Options dialog box appears.
2. Select the appropriate options in the right-hand pane, then click OK.

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 connected to a machine that is running the MetaFrame server, log into the remote system and run applications. Alternatively, 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. Test the environment using the Window Citrix Client to start a Citrix session.

Testing in load-balanced environments

If testing an environment that includes a server farm, 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. 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.

When ICA files are not provided by an application, specify connection information using the Published Applications or the Single Server options in the Citrix Record Options dialog box. These options also enable you to specify lists of published applications and servers to invoke when you log on to the Citrix client session.

Testing using the Citrix Web Interface

When you initiate Citrix-published applications using a web browser, you can capture and playback scripts in QALoad using the Citrix Web Interface. The Citrix Web Interface starts a Universal session with the Citrix and WWW middlewares selected. The WWW middleware intercepts the ICA file created by the web server. It passes data to and from the Citrix Server during the capture process, enabling you to test the performance of applications that are available through internet connections.

About the Citrix Web Interface

Capture and playback scripts for Citrix applications accessed through a web browser using the Citrix Web Interface. This starts a Universal session for both the Citrix and WWW middlewares, which enables the WWW middleware to pass information to and from the Citrix server.

When selecting a published application through a web browser, the request is sent to the web server. The web server creates an ICA file for the requested application and returns it to the web browser. When the web browser finds an associated application to handle the ICA file, the Citrix client starts the published application requested. The ICA file used during the capture process is saved in `QALoad\Middlewares\Temp`.

Note: Only the first ICA file received by the web browser is recorded.

The converted script contains both Citrix and WWW functions. A new variable, `char* strICAFileName[N]`, is declared at the top of the script. All allocated memory is released at the end of the script.

Accessing a Citrix Session

Use one of the following methods to begin recording a Citrix session.

To access a Citrix Client session:

- From the Session menu, click Citrix>Windows Client.

  OR

- On the toolbar, select the down arrow next to the Citrix button on the toolbar, then select Windows Client.

To access the Citrix Web Interface when selecting a Citrix session:

- From the Session menu, click Citrix>Web Interface.

  OR

- On the toolbar, click the down arrow next to the Citrix button, then select Web Interface.

The Universal session starts with both the Citrix and WWW middlewares selected. In the Citrix tab, Web Interface is selected in the Type field.

To access the Citrix Web Interface from a Universal session:

1. On the toolbar, click Session>Universal or click the Universal button on the toolbar.
2. Click Options>Record. The Universal Record Options dialog box appears.
3. In the Middleware Selection pane, select WWW, then select Citrix. The Universal Record dialog box displays both a WWW tab and a Citrix tab. In the Citrix tab, Web Interface is selected in the Type field.

Recording a Citrix Session

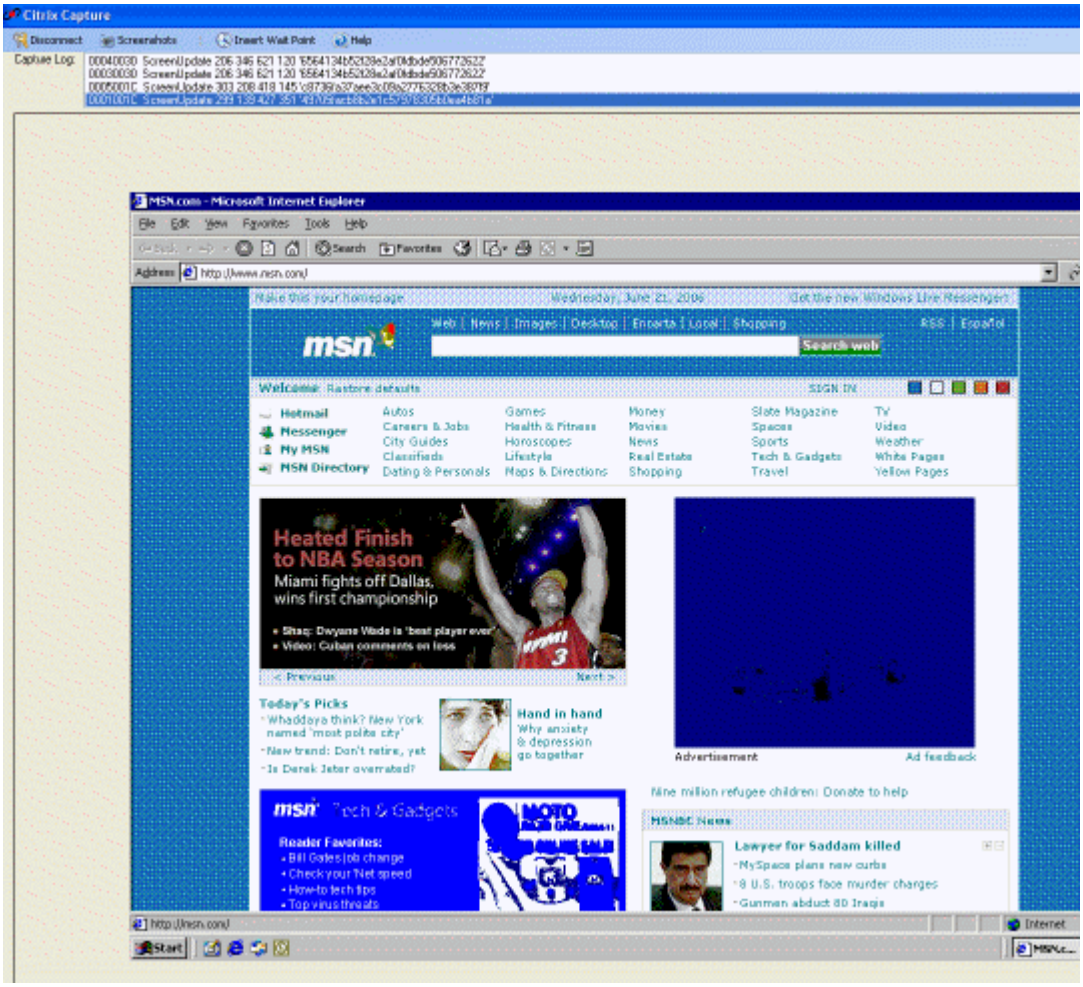
To begin recording a Citrix session, select the down arrow next to the Citrix button on the toolbar, then do one of the following procedures below.

Tips: Uppercase characters are not captured when the CAPS Lock key is on. Manually modify the script to use uppercase characters or hold down the SHIFT key during recording.
Similarly, the Windows Logo key is not supported. Do not use the Windows Logo key to start applications while recording a Citrix script.

To begin recording a Citrix Client session:

1. Select Windows Client to activate a new Citrix session.
2. Click Options>Record on the menu bar, and select the appropriate options.
3. Click Record on the Session toolbar.

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.



To begin recording using the Citrix Web Interface:

1. Select Web Interface to activate Universal session for the Citrix Web Interface.
2. Click Record on the Session toolbar. The Universal Record Options dialog box displays.
3. Select the appropriate options, then click Start Record.

Setting Citrix Convert Options

To set conversion options for Citrix Client session:

1. Choose Options>Convert in the Script Development Workbench menu bar. The Session Options dialog box appears.
2. Select the appropriate options, then click OK.

To set conversion options for Citrix Web Interface session:

1. Choose Options>Convert in the Script Development Workbench toolbar. The Session Options dialog box appears with both Citrix and WWW selected.
2. Click the Session Options tab.
3. Under Convert Options, click Citrix Convert Options.
4. Select the appropriate options, then click OK.

   Note: During the conversion process for both Client and Web Interface sessions, calls to the `CtxSetWindowTitle` method are sometimes placed at an incorrect line in the script. Determine the correct line by inspecting the capture file and moving the call to that position in the script.

   The `CtxSetWindowTitle` method is added by QALoad during conversion and should otherwise not be modified or manually added to a script.

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.

Inserting Screenshots in the Citrix Script

During the Citrix session, you can take a screenshot of the contents of the connection window. This inserts a wait point in the script for the screenshot you select and saves the image as a bitmap. You can select:

- Insert Wait for Full Screenshot - Inserts a wait point based on a full screenshot of the current display.
- Insert Wait for Partial Screenshot - Inserts a wait point based on a partial screenshot of the current display.
- Save Current Screenshot - Saves a screenshot of the current display to a bitmap file.

Note: Images are saved to `\ QALoad \Middlewares\Citrix\Captures\screenshot.`

To insert a full screenshot into the Citrix script:

1. In the Citrix Capture dialog box, click Screenshots>Insert Wait for Full Screenshot. The Screen Capture Preview screen displays the image.
2. Click Finish to confirm the screenshot. The Specify Screenshot Name dialog box displays with a list of all previously saved screenshots.
3. In the Name field, type a descriptive name for the image.

4. Click OK. The image is saved as a bitmap and a wait point for the screenshot is inserted into the script.

To insert a partial screenshot into the Citrix script:

1. In the Citrix Capture dialog box, click Screenshots>Insert Wait for Partial Screenshot. The selected screen displays with the message "Select a rectangular region of the screenshot you want to insert."
2. Hold down the left mouse button and drag the cursor across the screen to select the region to save. The Screen Capture Preview screen appears with the image you selected.
3. Click Finish. The Specify Screenshot Name dialog box appears with a list of all previously saved screenshots.
4. In the Name field, type a descriptive name for the image.
5. Click OK. The image is saved as a bitmap and a wait point for the screenshot is inserted into the script.

To save the current screen:

1. In the Citrix Capture dialog box, click Screenshots>Save Current Screenshot. The Screen Capture Preview screen displays.
2. Click Finish. The Specify Screenshot Name dialog box displays with a list of all previously saved screenshots.
3. In the Name field, type a descriptive name for the image.
4. Click OK. The image is saved as a bitmap.

Clearing Events from the Internal Queue

In response to mouse and keyboard input, the Citrix server sends screen update events of the updated screen image to the Citrix client application. You can synchronize these screen update events with the text or graphic updates in the application by using CtxWaitForScreenUpdate to insert waitpoints in the Citrix script.

When numerous matching screen update events occur, such as the flashing edit cursor, they stay in the internal queue indefinitely. This can cause the CtxWaitForScreenUpdate function to complete prematurely by matching events with previously stored screen updates. You can clear the internal queue by using the CtxScreenEventExists function in a simple loop. This flushes all past screen update events from the internal queue, so a following CtxWaitForScreenUpdate can successfully wait for the next occurrence of the desired screen update event.

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. 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.

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. 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;

.
.
.
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();
```

> 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.

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 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 CtxSetWindowMatchTitle 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 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();
}
```

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

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 QALoad enforces those requirements. For example, Java file names cannot contain spaces. If you try to include a space in your file name, QALoad gives 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. QALoad 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 QALoad'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.

Setting Classpaths in QALoad Player for EasyScript for Java

When loading a class by name in the QALoad Player during runtime, do not use a class name on the Enterprise Java Bean (EJB) InitialContext call. Instead, use a class instance or add a line of code before the JNDI lookup call. Refer to the following examples:

Using a class instance

Replace the InitialContext properties and JNDI names in the following example with the values that are appropriate for your application.

Before:

```
java.util.Hashtable ht = new java.util.Hashtable();
ht.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
ht.put(javax.naming.Context.PROVIDER_URL, "fh12623:1099");
javax.naming.InitialContext ic = new javax.naming.InitialContext(ht);
Object ref = ic.lookup("AddressMSvc");
```

After:

```
org.jnp.interfaces.NamingContextFactory nf = new
org.jnp.interfaces.NamingContextFactory();
java.util.Hashtable ht = new java.util.Hashtable();
ht.put(javax.naming.Context.PROVIDER_URL, "fh12623:1099");
javax.naming.Context ic = nf.getInitialContext(ht);
Object ref = ic.lookup("AddressMSvc");
```

Adding a line of code before the JNDI lookup call

Add the following line before the JNDI lookup call:

```
Thread.currentThread().setContextClassLoader(getClass().getClassLoader());
```

If it is a static method, use the following sample, replacing CLASSNAME with the class name of the code.

```
Thread.currentThread().setContextClassLoader(CLASSNAME.class.getClassLoader());
```

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();
```

Oracle

Recording an Oracle Session

Click the Oracle button on the toolbar to open an Oracle session.

To set Oracle recording options and begin recording:

1. Click Options>Record on the menu bar. The Session Options dialog box appears.
2. Select the appropriate options in the right-hand pane, then click OK.
3. Click the Record button on the Session toolbar.

Setting Oracle Convert Options

To set conversion options for an Oracle session:

1. Choose Options>Convert in the Script Development Workbench menu bar. The Session Options dialog box appears.
2. Select the appropriate options in the right-hand pane, then click OK.

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 QALoad 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, identifies 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 opens, listing the value used in each file.
5. If you do not 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

Setting Oracle Forms Convert Options

Use the following procedures to set Oracle Forms convert options and advanced convert options. Use the advanced convert options to customize the post-capture steps taken before converting the capture file to a script, or to manually perform post-capture processing of a proxycap (hexadecimal encoded) file. This produces a new cap file, postcapweb file, longcap file, and sortedcap. This process overwrites these files when they already exist.

To set conversion options for Oracle Forms in a single session:

1. Choose Options>Convert in the Script Development Workbench toolbar. The Session Options dialog box appears.
2. Select the appropriate options, then click OK.

To set conversion options for Oracle Forms in a Universal session:

1. Choose Options>Convert in the Script Development Workbench toolbar. The Session Options dialog box appears with both Oracle Forms Server and WWW selected.
2. Click the Session Options tab.
3. Under Convert Options in the tree view, click Oracle Forms Convert Options.
4. Select the appropriate options, then click OK.

To set Advanced convert options:

1. Choose Options>Convert in the Script Development Workbench toolbar to open the Session Options dialog box.
2. In the Convert Options tree view, click Advanced Convert Options.

3. Select the appropriate options, then click OK.

Recording Oracle Forms Server Sessions

You can record an Oracle Forms session in a single session, or in a Universal session for Oracle E-Business Suite 11i (EBS-11i) and Oracle E-Business Suite 12 (EBS-12). Select the down arrow next to the Oracle Forms Server Session button on the toolbar, then follow one of the procedures below.

To begin recording an OFS single session:

1. Select Oracle Forms to activate a new OFS single session.
2. From the toolbar, select Options>Record, and select the appropriate options.
3. Click Record on the Session toolbar.

To begin recording OFS in a Universal session:

1. Select E-Business Suite 11i or E-Business Suite 12 to activate Universal session for the Oracle Forms and WWW middlewares.
2. Click Record on the Session toolbar. The Session Options dialog box displays with the Middleware tab on top. Both WWW and Oracle Forms Server are selected.
3. Click the Session Options tab.
4. In the tree view under Record Options, select Oracle Forms Record Options. The Oracle Forms Record Options pane displays on the right. The session you initiated, either E-Business Suite 11i or E-Business Suite 12, automatically displays in the Forms Environment field.
5. Select the appropriate options, then click OK to begin recording.

Note: When you record EBS-12 using Internet Explorer 6, changing the Accessibility option on the EBS-12 login page from the default (None) requires modifications to your script in order for it to run properly. Refer to Changing Accessibility Options in Oracle EB-12 for more information.

Oracle Forms Recording Modes

QALoad supports recording Oracle E-Business Suite (EBS) 12 and 11i, and Oracle Applications using Forms 10g, 9i, and patched 6i (versions 6.0.8.14 and up). These applications may be recorded in HTTP mode (also called Servlet mode), SSL mode (also called HTTPS or Secure Servlet mode), and socket mode. These recording types are described briefly below.

Recording Servlet Mode

Oracle Forms Applications use HTTP to send Forms data across the network. To record in Servlet mode, select Servlet in the Connection Mode list 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. Server-side recording is not available for EBS-12 or EBS-11i.

Recording SSL Mode

To record an OFS application in Secure Servlet (SSL) mode, select Secure Servlet in the Connection Mode list in the Record Options dialog box. For non-EBS-12 applications, you must specify a certDB file by entering 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. This field is not required for EBS-12.

Note: SSL mode is not available with server-side 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.

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 Application Server and Oracle E-Business Suite 11i 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 all Oracle E-Business Suite and Oracle Applications 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. QALoad automatically extracts ICX tickets and any necessary cookies from the WWW middleware traffic and passes them to the OFS middleware.

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 Record Options dialog box. Forms connection modes include server-side recording, HTTP, HTTPS, or socket.

Server-side recording is limited to applications that use Oracle Application Server. 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.

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 Oracle Forms Record 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, QALoad 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.

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, QALoad 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, QALoad wraps the binary data inside an HTTP stream and invokes Java's HTTP calls.
- For socket mode, QALoad 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.

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.

Verifying OFS Window Creation

The ofsWindowCreated command does two things. First, it ensures that for the ofsSendRecv() command preceding it, the server includes in the response that a FormWindow object with the title specified is created. The script returns FALSE if the object is not created. Second, the original (captured) control ID is associated with the object representing the control in OFS replay's internal list of controls, which is in addition to the (runtime) control ID that is tracked.

This allows better tracking of FormWindow GUI controls that exist at runtime by being able to match up both the control name and the original control ID.

The ofsWindowCreated command only handles objects of type FormWindow.

Using the ofsWindowCreated Script Command

The following example shows an example from a modified OFS script:

```
   ofsSendRecv(1); //ClientSeqNo=1|MsgCount=1|1137439721.484

   if (!ofsWindowCreated(11, "WINDOW_LARGE_GRAPH"))
{
  // Only fail if this window was not created
  RR__FailedMsg(s_info, "WINDOW_LARGE_GRAPH was not created!");
}
ofsWindowCreated(15, "WINDOW_START_APP");
ofsWindowCreated(20, "WINDOW_CONTROLSTEST");
ofsWindowCreated(24, "WINDOW_GRAPHICSTEST");
ofsWindowCreated(28, "WINDOW_DATABASETEST");
```

SAP

Overview of SAP

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

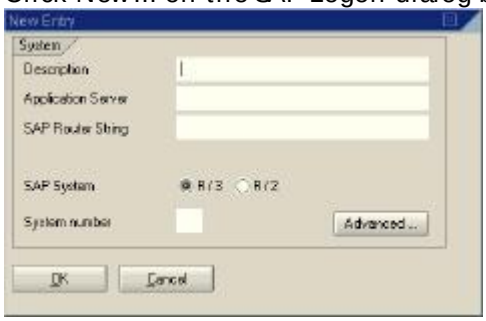
Once you have connected to a machine that is running the SAP server, you can log on and interact with the SAP applications.

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 New... 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)

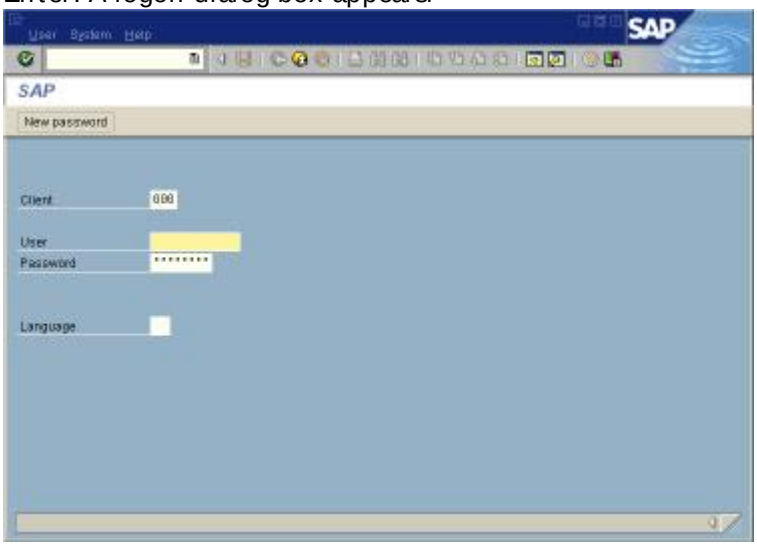
Save Server Description: Select to specify and save the server description (name) to which you want to connect during recording. If this check box is not selected, you are prompted for a server description during the log on process.

Recording an SAP Session

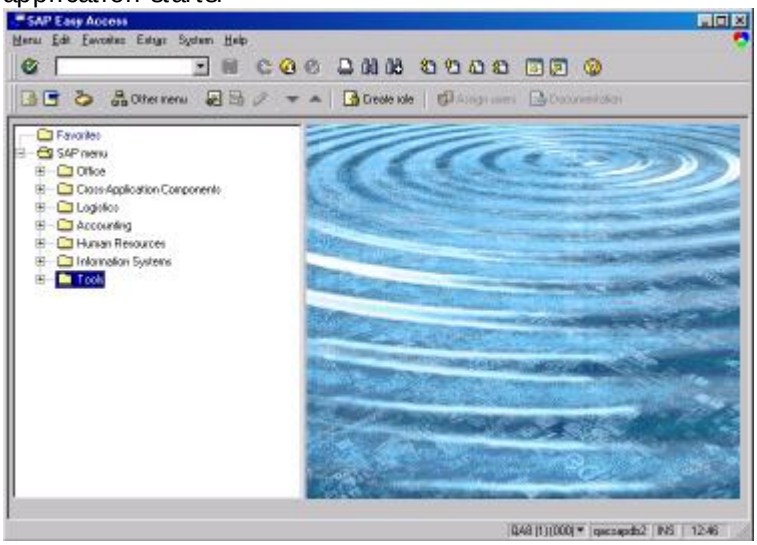
An SAP server connection must be configured before you can connect with QALoad. See Configuring an SAP Client for Load Testing for more information. Additionally, your SAP administrator must set the `SAPGUI/User_Scripting` security profile parameter to TRUE to successfully record a script. For more information about SAP security settings, refer to the SAP publication titled "Sapgui Scripting Security".

To record an SAP session:

1. If you have not already chosen SAP as the session type, click SAP Session to activate a new SAP session.
2. In the Script Development Workbench, click Record on the Session toolbar.
3. 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 logon dialog box appears.



4. Type a user ID in the User field and the password in the Password field. Press Enter. The SAP application starts.



5. In the SAP application, turn off the scripting and notification options. Click Customizing of local layout and choose Options. The Options dialog box appears. On the Scripting tab, select Enable Scripting and clear the two Notify check boxes.
6. Begin recording actions in SAP.

SAP Convert Options

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.

Insert SAP control comments: Select one of the following to insert SAP control lists as comments in your script.

- None: There is no control information shown as a comment in the script.
- Textfields and Statusbars: Control types GuiTextField or GuiStatusbar will be in the script as comments with information: type, name, and id string.
- All controls: All controls will be in the script as comments with information: type, name, and id string.

Build SAP Libraries: Click Build 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 Command Reference (Versions 6.x)

QALoad provides descriptions and examples of the various commands available for an SAP script. For details, refer to SAP 6.x Language Reference Commands.

Viewing the SAP Control Log

Assist your scripting by using the SAP capture control log file to view the information that was present at capture . The log file is in XML format.

To open the control log:

1. In the Script Development Workbench Workspace, click the Captures tab or the Scripts tab, then right-click on an SAP script or capture file.

   OR

   Open an SAP script or capture file, then right-click on the open file.
2. From the right-click menu, choose View Control Log.

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

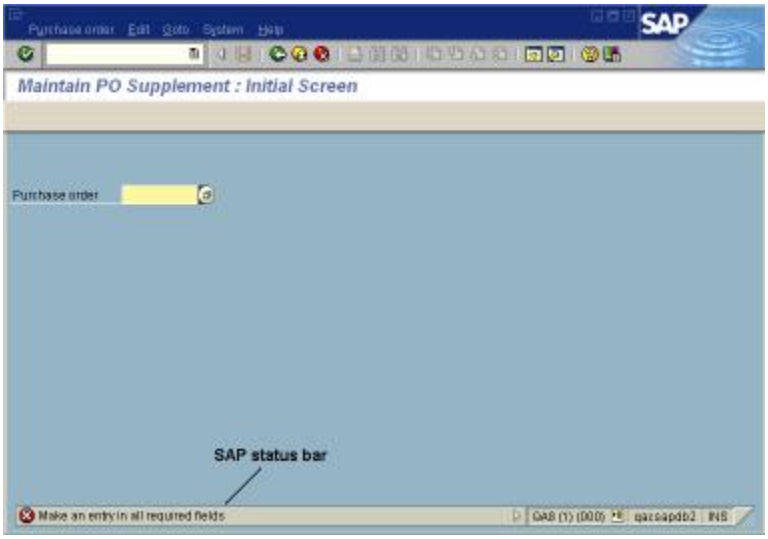
You may need to modify your script to handle multiple logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this 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 want 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:
   a. If the character is "\r", "\n", "\t", or "\f", it is represented in the script as a normal C escape character.
   b. 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.
   c. 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();
}
```

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

Converting a WWW Script

QALoad has been enhanced to enable you to control how much automated processing is performed during test execution by choosing HTML mode or HTTP mode for script conversion. When going through the WWW convert for the first time, you are now prompted to choose between HTML mode and HTTP mode.

Tip: Compuware recommends that you use the WWW Convert Options dialog box to convert between modes, rather than manually edit the script.

Typical HTML processing uses more processor cycles and memory during test execution, but requires less scripting. The Visual Navigator is generated using QALoad's Document Object Mode (DOM) approach, which also is used for playback.

HTTP processing instructs QALoad to disable most of its automatic HTML handling engine during replay, and it also affects how the script is generated. In this mode, the convert process uses some HTML parsing to generate the Visual Navigator. This parsing method also is used for playback instead of the DOM. The HTTP mode replaces Minimized Processor mode. Scripts converted before QALoad release 5.6 using Minimized Processor mode will replay in HTTP mode.

Caution: Once you convert a script using HTTP mode or HTML mode, you must reconvert the script to change the convert mode. This is necessary so that the convert and replay methods match. Failure to change modes by reconverting the script can cause the script to replay incorrectly.

Using the WWW Record Options

Use the WWW Record Options to determine how QALoad handles captures web page transaction data.

To set the WWW recording options:

1. With a WWW session open, click Options>Record. The Session Options dialog box appears with the WWW Record Options highlighted in the tree view control. The WWW Record Options dialog box appears on the right.
2. Select the appropriate options, then do one of the following:
   - Click OK to save the selections and return to the WWW session screen.
   - Click Advanced Record Options in the tree view control. The Advanced Recording Options dialog box appears on the right.
3. Select the appropriate options.

4. Click OK. The basic record options and the advanced options are saved and you return to the WWW session screen.

Using the WWW Convert Options

Use the WWW Convert Options to control how QALoad produces scripts from recorded transaction data. You can also choose the convert and playback mode for WWW middleware sessions. The convert and playback mode determines the amount of memory and processor QALoad uses during replay.

To select the processing mode:

1. With the WWW session open, click Options>Convert. The Session Options dialog box displays.
2. Click WWW Convert Options in the tree view control. The WWW Convert Options dialog box appears on the right side of the screen.
3. Select the convert mode to use for the WWW middleware sessions. You can choose:
   - HTML Mode - to reduce the amount of scripting.
   - HTTP Mode - to control processor and memory usage.

Tip: Compuware recommends that you use the WWW Convert Options dialog box to convert between modes, rather than manually edit the script.

4. Click each of the following to display the associated dialog box and select the appropriate options:

Note: The options available for Parsing depend on whether you select HTTP or HTML mode.

- Parsing
- Verification
- Caching
- Traffic Filters
- Rule Filters
- Connection Settings
- Content Type Handling
- General
- Siebel

5. Click OK to save the settings and return to the main WWW session screen.

Caution: Once you convert a script using HTTP mode or HTML mode, you must reconvert the script to change the convert mode. This is necessary so that the convert and replay methods match. Failure to change modes by reconverting the script can cause the script to replay incorrectly.

Using the WWW Playback Options

The WWW Playback options you select affect the entire script.

To select the WWW Playback Options:

1. With a script open in a WWW session, double-click Playback Options in the tree view. The available session options display in the tree view.

2. Double-click WWW Playback Options in the tree view.
3. Click each playback option in the tree view to display the associated dialog box on the right.
4. Click each of the following to display the associated dialog box and select the appropriate options:

Note: The options available for Parsing depend on the mode you select in the WWW Convert Options dialog box.

- Parsing (HTML Mode or HTML Mode)
- Caching
- Traffic Filters
- Connection Settings
- Proxy
- General
- Siebel

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.

Best Practices

Comparison of HTTP Mode and HTML Mode

Choosing HTML mode or HTTP mode for script conversion enables you to control how much automated processing is performed during test execution. Typical HTML processing uses more processor cycles and memory during test execution, but requires less scripting. HTTP processing instructs QALoad to disable most of its automatic HTML handling engine during replay, and it also affects how the script is generated.

Features Available for Selection	HTTP Mode	HTML Mode	Convert Options Page
Parsing	Yes	No (Always On)	Parsing
Dynamic Cookie Handling	Yes	No (Always On)	Parsing

Dynamic Redirects	Yes	No (Always On)	Parsing
Authentication	Yes	Yes	N/A
HTTPS	Yes	Yes	N/A
Extracts	Yes	Yes	N/A
Title Verification	Yes	Yes	Verification
Content Check	Yes	Yes	Content Type Handling
Chunked Data	Yes	Yes	N/A
Streaming Media	Yes	Yes	General
Traffic Filters	Yes	Yes	Traffic Filters
Parameterization Rules	Yes	Yes	Rule Filters
Allow JavaScript Execution	Yes	No	Parsing
Click_Ons	No	Yes	N/A
Fill In Forms	No	Yes	N/A
Caching	Yes	Yes	Caching
Expert User	Yes	Yes	N/A
Universal Mode	Yes	Yes	N/A
Represent CJK as Octal Characters	Yes	Yes	General
UTF-8	Yes	Yes	N/A
XML Requests	Yes	Yes	General
Meta Refresh	Yes	Yes	General
Automatic Sub-request Processing	Yes	No (Always On)	Parsing
Set JavaScript Execution Level	No	Yes	Parsing
Set JavaScript Loop Timeout	Yes	Yes	Parsing
Manually Select Sub-requests	Yes	Yes	Parsing

Persistent Connections During Replay	Yes	Yes	Connection Settings
Graceful Socket Shutdown	Yes	Yes	Connection Settings
Baud Rate Emulation	Yes	Yes	Connection Settings
Reuse SSL Session ID	Yes	Yes	Connection Settings
Siebel Support	Yes	Yes	Siebel
Strip All Cookies	Yes	Yes	General

Using HTML Mode

HTML mode converts and replays scripts using full Document Object Model (DOM) and JavaScript engine. HTML documents are fully parsed in HTML mode. JavaScript can be executed at different levels, including NONE, LIMITED, and FULL. Because HTML mode uses full DOM and JavaScript engine, it uses more processor cycles and memory during test execution, but requires less scripting.

Use HTML mode to create load testing scripts quickly and easily, and for load tests requiring fewer virtual users (VU) running on any player machine. You can use HTML mode to create scripts for heavier load testing, but the scripts may require more player machines on which to run the VUs.

Considerations

- Javascript - Javascript can be executed at different levels, including None, Limited, and Full.
- CPU usage - Uses more processor cycles during test execution
- Memory - Uses more memory during test execution
- Parameterization - Requires less scripting and less parameterization than HTTP mode because QALoad is better able to infer user actions using its DOM.

Use this mode if you are:

- testing a web site that uses complex JavaScript with complex HTML pages that contains a significant number of links, tables or HTML objects
- a novice tester and wish to work with scripts that more closely resemble the user's interactions with the application under test
- only testing with a low number of virtual users or have significant hardware resources available for load testing execution

Tip: Compuware recommends that you use the WWW Convert Options dialog box to convert between modes, rather than edit the script manually.

Sample Script

The following shows a sample of a script with HTML mode selected:

```
...
...
SET_SCRIPT_LANGUAGE(SLID_English);
DO_WWWInitialize(s_info, HTML_MODE);
```

```
...
...
BEGIN_TRANSACTION();
...
...
```

Using HTTP Mode

HTTP mode converts and replays scripts using a simplified HTML parser and limited JavaScript engine. You can control the level of HTML document parsing, and whether JavaScript should be executed. HTTP mode does less processing during replay and uses less memory because it does not create Document Object Models (DOM) for each processed HTML page.

HTTP mode may require more parameterization than HTML mode. During conversion, actions are represented in the visual script by NavigateTo or PostTo tree items instead of ClickOn or Fill-In Form tree items. This means that any dynamic variables passed from reply to request must be managed in the script. Use the Variable Replacement Wizard to develop rules to find parameters in your script and replace them with variables.

Considerations

- JavaScript — Because JavaScript execution is limited, some scripts may require customization to emulate Javascript operations not executed by QALoad.
- Automatic Sub-requests — When Parsing is selected, automatic Sub-requests are enabled. When not enabled, QALoad executes the same sub-requests made during the capture phase (instead of parsing the server responses). This makes playback highly scalable, but very dynamic web applications may require additional scripting. All subrequests and additional subrequests are selected by default during the convert.
- Functions — Converted scripts use only the Navigate_To() and Post_To functions; the Click_On() function is disabled.
- Parameterization — May require more parameterization than HTML mode.
- Memory and CPU Usage — Does less processing and uses less memory during replay.
- Dynamic variables — Dynamic variables passed from reply to request must be managed in the script.

Use this mode if you are:

- testing a web site that makes little use of Javascript and creates pages with few HTML options on them
- an experienced load tester and wish to work with scripts that more closely resemble the applications HTTP interactions with the server under test
- you are testing for high virtual user throughput or simulating high numbers of concurrent virtual users from limited hardware resources

Tip: Compuware recommends that you use the WWW Convert Options dialog box to convert between modes, rather than edit the script manually.

Sample Script

The following shows a sample script with HTTP mode selected:

```
...
...
SET_SCRIPT_LANGUAGE(SLID_English);
```

```
DO_WWWInitialize(s_info, HTTP_MODE);
...
...
BEGIN_TRANSACTION();
...
...
```

Sample Scripts for Convert Options

Below are samples of how the WWW convert options appear in a script. Compuware recommends that you use the WWW Convert Options dialog box to convert between modes, rather than edit the script manually.

Once you convert a script using HTTP mode or HTML mode, you must reconvert the script to change the convert mode. This is necessary so that the convert and replay methods match. Failure to change modes by reconverting the script can cause the script to replay incorrectly.

Parsing

Parse Pages

When the Page Parsing option is selected on the WWW Convert Options, QALoad performs limited parsing operations on incoming HTML pages. This option only applies to HTTP mode, and is disabled if HTML mode is selected as the convert mode.

You should disable this option if you find that you need to fit more HTTP mode virtual users on a single player machine. Disabling this option can decrease the processor usage required by a virtual user.

Considerations

- When selected, limited parsing is performed on incoming HTML pages.
- Must be selected to enable the Allow Javascript execution option.
- Must be selected to enable the Automatically process sub-requests option.
- May increase processor usage required by a virtual user.

Use this option to:

- Allow Javascript to be executed in HTTP mode.
- Allow automatic sub-requesting to be used in HTTP mode.
- Enable document verification in HTTP mode.

Allow JavaScript Execution

This option is only enabled when the Parse Pages option is selected in HTTP conversion mode. When this option is enabled, QALoad executes JavaScript in replay as if the LIMITED JavaScript Execution Level is used. Since JavaScript in HTML mode is controlled by the JavaScript Execution Level option, this option is disabled if HTML mode is selected as the convert mode.

Considerations

- The option to turn on or turn off JavaScript execution is available only in HTTP mode. JavaScript execution in HTML mode is controlled by the JavaScript Execution Level option.

- When enabled, QALoad executes JavaScript at the Limited execution level.
- May decrease the amount of parameterization required for certain scripts.
- May increase the processor time required by a virtual user.

Use this option to:

Decrease the amount of parameterization required.

Automatically Process Sub-requests in HTTP Mode

When this option is selected, QALoad does not insert sub-requests (such as .jpg, .gif, .css, .js, etc.) directly into the script file, but automatically makes these requests during replay by parsing them out of the HTML pages.

This option is only enabled when HTTP mode is selected and the Parse Pages option is enabled. HTML mode always processes sub-requests automatically.

Considerations

- When this option is disabled, QALoad executes the same sub-requests made during the capture phase instead of parsing the server response.
- Very dynamic web applications may require additional scripting.
- All sub-requests and additional sub-requests are selected by default during convert.
- Enabling this option requires more processing during playback.

Use this option to:

- Allow QALoad to manage sub-requests.
- Reduce the length of .cpp script files.
- Increase .cpp script file readability.

Dynamic Redirect Handling

Selecting this option enables the QALoad replay engine to dynamically handle 3XX redirects. When a 3XX redirect page is returned during replay, QALoad processes the redirect and requests the redirected page as well. This option only applies to HTTP mode, and is disabled if HTML mode is selected as the convert mode.

Considerations

- This option should normally only be disabled if you do not want the script to follow 3XX redirects.
- Dynamic redirect handling does not handle JavaScript redirects.
- Dynamic redirect handling does not handle meta refresh redirects.

Use this option to:

- Dynamically handle 3XX redirects.
- Process the redirect and also request the redirected page.

Dynamic Cookie Handling

When you select this option, QALoad automatically processes dynamic cookies during replay. This option only applies to HTTP mode, and is disabled if HTML mode is selected as the convert mode. QALoad always handles dynamic cookies in HTML mode.

Considerations

- Not selecting this option hard codes cookie values from the capture file into the script file.
- When this option is not selected in HTTP mode, QALoad places Set(NEXT_REQUEST, COOKIE) commands into the generated script file.
- When this option is enabled, and you still can insert Set(NEXT_REQUEST_ONLY, COOKIE) statements. This can override the cookie value that QALoad sends for any cookie that is being dynamically handled.
- You should only disable this option if you require more control over the cookies that QALoad sends during replay.

Use this option to:

- Ensure that dynamic cookies are processed during replay.
- Ensure that dynamic cookies are not automatically inserted into the script during convert.

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 “Handle Dynamic Cookies” check box selected.
...
...
Set(EVERY_REQUEST, HTTP_MODE_HANDLE_DYNAMIC_COOKIES, TRUE);
BEGIN_TRANSACTION();
...
...
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
...
...
END_TRANSACTION();
...
...
```

Script Example with the Dynamic Cookie Handling Option Selected

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

```
...
...
Set(EVERY_REQUEST, HTTP_MODE_HANDLE_DYNAMIC_COOKIES, TRUE);
BEGIN_TRANSACTION();
...
...
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
...
```

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

Script Example with the Dynamic Cookie Handling Option Not Selected

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

```
...
...
Set(EVERY_REQUEST, HTTP_MODE_HANDLE_DYNAMIC_COOKIES, FALSE);
BEGIN_TRANSACTION();
...
...
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
/* Request: 2 */
Set(NEXT_REQUEST_ONLY, COOKIE, "SaneID",
"172.22.24.180-4728804960004");
Set(NEXT_REQUEST_ONLY, COOKIE, " SITESERVER ",
" ID=f0544199a6c5970a7d087775f83b23af ");
Navigate_To("http://www.host.com/cgi-bin/cookies5.pl ");
...
...
END_TRANSACTION();
...
...
```

JavaScript Execution Level

This option controls the types of JavaScript statements that QALoad executes while running in HTML mode. The levels are None, Limited, and Full. Limited is selected by default.

This option only applies when HTML is selected as the convert mode. In HTTP mode, this option is disabled and there is only a single level of JavaScript execution.

The following table explains each JavaScript Execution Level:

Execution Level	Description
None	No JavaScript is executed during replay.
Limited	Only the following JavaScript statements are executed during replay: ! document.cookie = ! window.document.cookie = ! document.write() ! window.document.write() Images are also requested where the image.src statement occurs.
Full	All JavaScript statements are executed during replay.

Considerations

- When using the Limited JavaScript Execution Level, you may need to do further parameterization and edit the script.
- In HTTP mode, only the Limited level of JavaScript execution is available.
- Using no JavaScript execution or a Limited level execution requires less processing time and memory usage than the Full level of JavaScript execution.

Use this option to:

Control the amount of processor and memory usage.

JavaScript Loop Timeout

This option controls the amount of time that a JavaScript runs before QALoad terminates the JavaScript code. JavaScript can run into infinite loops, or take a very long time to run. This option applies to both HTML and HTTP mode.

Considerations

- The timeout value should be decreased if you find that an unnecessary JavaScript is running longer than the timeout value.
- The timeout value should be increased if you find that a necessary JavaScript is running longer than the timeout value.
- Make sure that a necessary JavaScript is able to complete before the timeout value of time is reached.

Use this option to:

Prevent JavaScripts from running into infinite loops or from running indefinitely .

Connection Settings

Connection Settings

These options control how QALoad handles connections to target servers during replay.

Considerations

- Connection Options — Select to keep the connection to the server open for each request sent to the server.
- Graceful Socket Shutdown — Select to control how QALoad terminates its socket connections.
- Baud Rate Emulation — Select to simulate a specific baud rate for transmission of requests.
- SSL Session — Select to instruct QALoad to reuse the current session's communication information (session ID) for all page requests within the transaction.

Use this option to:

Control how QALoad handles connections to target servers during replay.

Persistent Connections During Replay

This is an option placed in the script to be used at replay time. Selecting this option keeps the connection to the server open for each request sent to the server.

Considerations

Keeping connections open (persistent) can increase the speed at which requests are sent to the server.

Use this option to:

Control processor and memory usage for requests.

See also:

Sample Scripts: Persistent Connections During Replay

Connection Settings

Max Concurrent Connections

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

Considerations

- Keeping the maximum number of connections open can increase the speed at which requests are sent to the server.
- In HTTP mode, if the Automatically process subrequests option on the Parsing Options page is NOT selected, this option is disabled.

Use this option to:

Control processor and memory usage for requests.

Max Connection Retries

This option specifies the number of times during replay that QALoad will attempt to connect to the server after timing out.

Considerations

You must select Persistent connections during replay to enable this option.

Use this option to:

Control processor and memory usage for requests.

Server Response Timeout

This option specifies, in seconds, the length of time during replay that QALoad waits for data from the server before timing out.

Considerations

You must select Persistent connections during replay to enable this option.

Use this option to:

Control processor and memory usage for requests.

Graceful Socket Shutdown

When this option is enabled, QALoad attempts to gracefully terminate socket connections by making sure that all remaining data has been sent and received before closing a socket connection.

Considerations

Enabling this option usually requires more time before the socket is shut down.

Use this option to:

Control how QALoad terminates its socket connections.

Baud Rate Emulation

Use this option 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.

Considerations

- The DO_SetBaudRate command is inserted into the script with the specified baud rate as its only parameter.
- If baud rate emulation must be asymmetric (the 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.

Use this option to:

- Simulate a specific baud rate for transmission of requests.
- Simulate a specific baud rate for reception of requests.

General

Proxy HTTP Version

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

Consideration

HTTP 1.1 requests receive chunked replies.

Use this option to:

Specify the HTTP version of requests and sub-requests sent to the server.

META Refresh Threshold

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">).

Considerations

- If the 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 value you specify, the page is treated as a regular page.

Use this option to:

Avoid infinite loops in the script. Infinite loops can occur if a page refreshes periodically to update data.

Sample Scripts

Script Example with Option Selected

Script Example with Option Not Selected

Script Example with the META Refresh Threshold Option Selected

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

```
. . .
...
Set(EVERY_REQUEST, HTTP_VERSION, "1.1");
Set(EVERY_REQUEST, META_REFRESH_THRESHOLD, 5);
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
Navigate_To(“http://host/path/to/page.pl");
Verify(PAGE_TITLE , "You have reached the final page!!");
```

Script Example with META Refresh Threshold Option Not Selected

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

```
. . .
...
// WWW General Options
Set(EVERY_REQUEST, HTTP_VERSION, "1.1");
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
Navigate_To(“http://host/path/to/page.pl");
Verify(PAGE_TITLE, “Just Wait”);
DO_SLEEP(5);
/* Request: 2 */
Navigate_To(“http://host/path/to/realpage.pl");
Verify(PAGE_TITLE, "You have reached the final page!!");
```

Represent CJK as Octal Characters

When this option is selected, the double-byte characters used for Chinese, Japanese, and Korean (CJK) scripts are converted into octal format.

Considerations

- 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.
- Data stays in encoded format throughout the load test: from capture, through convert and replay, to the analysis of the timing file.

Use this option to:

- Enable the encoding of captured data from web applications containing Double Byte Character Sets (DBCS) such as Chinese, Japanese, or Korean.
- Encode all native characters when native character support cannot be used.

Streaming Media

QALoad supports two types of streaming media:

- RealOne Player
- Windows Media Player

Considerations

- Streaming media is not supported through firewalls and across proxies.
- 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. Simply 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.

Use this option:

For audio and video download testing of scripts with Windows Media Player or RealOne Player and their supported media formats through a WWW session.

Strip All Cookies from Request

When this option is selected, no cookies are sent with requests.

Use this option to:

Specify whether cookies are sent with requests.

Document Title Verification

Enables document title verification for main requests of type HTML or TEXT.

Considerations

There are three supported methods for verifying a document title: search for the Entire Document Title, the Prefix of the document title, or the Suffix of the document title.

Use this option to:

- Create scripts that verify characters contained in document titles.
- Detect and handle error messages that are returned in an HTML page.

Caching

When a caching option is enabled, requested images, .css, or .js files are cached at playback time.

Considerations

- Types of documents that QALoad caches on the virtual user during replay are: None, Images, CSS, JS, and Images, or All.
- The caching level names denote the types of documents that QALoad caches on the virtual user during replay.

- If these documents are referenced after the virtual user has already requested and received the document once, they are not requested from the target server again by that virtual user.

Use this option to:

Select the type of caching you would like the WWW replay engine to use during playback.

Script Samples

Caching Option: None

```
Set(EVERY_REQUEST, CACHING, NO_CACHING);
```

Caching Option: Images

```
Set(EVERY_REQUEST, CACHING, IMAGES);
```

Caching Option: CSS, JS, and Images

```
Set(EVERY_REQUEST, CACHING, IMAGES_JS_AND_CSS);
```

Caching Option: All

```
Set(EVERY_REQUEST, CACHING, ALL);
```

Traffic Filters

Traffic Filters allows you to filter out certain requests while playing back a script.

Considerations

Traffic filters do not affect XML requests that are written to the Visual Navigator tree when a capture file is converted.

Use this option to:

- Determine which traffic should be:
  - included in your script
  - blocked from your script
  - converted to subrequests in the script
- Also block specific substrings in the path of URLs, such as .jpg to block all JPEG images.

Rule Filters

Selects the Rule Folders that are scanned for matching rules when converting a WWW capture file into a Visual Navigator script.

Considerations

- Rules are sets of parameters that are established to substitute certain pre-established variables for system generated variables.
- Parameters and Rules are established using the Variable Replacement Wizard.

Use this option to:

Control the rules QALoad uses during the convert process to help parameterize scripts.

Enable Siebel Support

The Enable Siebel Support option allows QALoad to identify if the capture file to be converted is a Siebel file.

Considerations

- This option results in adding a new command in the script called SiebelInitialize, which identifies the script as a Siebel script.
- Two additional commands are also inserted into the script: SiebelUpdatePage and GetSiebelValue. These two commands are only added if there were HTML responses in the capture file, which resulted in generation of Siebel variables.
- When the convert process is complete, all HTML pages with Siebel variables attached to them at the convert process are output into the script file so playback can retrieve values from the Siebel Correlation Library. These parameters appear in various PostTo commands in the script and are correlated automatically to local variables created by the Siebel Correlation Library.

Note: When you want to modify the values of the parameters provided to create a new value for each transaction, you must manually parameterize the values.

Use this option to:

Record, modify, and play back scripts for the purpose of performance testing against Siebel applications and environments.

Script Samples

Example Script with Option Selected

Example Script with Option Not Selected

Script Example with the Enable Siebel Support Option Selected

The following example has the Enable Siebel Support check box selected.

```
CLoadString __Siebel_1 = "VRId-3";
CLoadString __Siebel_2 = "";
CLoadString __Siebel_3 = "SWERowId0=1-7BZ";
CLoadString __Siebel_4 = "";
CLoadString __Siebel_5 = "1-136";
CLoadString __Siebel_6 = "SWERowId0=1-7BZ";
CLoadString __Siebel_7 = "";
CLoadString __Siebel_8 = "1-1UY";
CLoadString __Siebel_9 = "";
CLoadString __Siebel_10 = "1-1UY";
CLoadString __Siebel_11 = "";
CLoadString __Siebel_12 = "1-1UY";
…
…
// WWW General Options
Set(EVERY_REQUEST, HTTP_VERSION, "1.1");
SiebelInitialize(); // Siebel support is enabled

….
….
Post_To("http://dtw-siebel78/callcenter_enu/start.swe");
// Get the Siebel parameters from this page
SiebelUpdatePage();
__Siebel_5 = GetSiebelValue(" S_BC3_S09_R06_FID ");
__Siebel_8 = GetSiebelValue(" S_BC3_S09_R05_FID ");
__Siebel_10 = GetSiebelValue(" S_BC3_S09_R05_FID ");
__Siebel_12 = GetSiebelValue(" S_BC3_S09_R05_FID ");
….
```

```
….
// Original string was:
//   "1-136"
// Variablized string is:
//   "{$ VAR:Siebel-5 $}"
//
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWERowId", __Siebel_5);
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEC", "7");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEMethod", "PositionOnRow");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEReqRowId", "1");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWERPC", "1");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEApplet", "Contact Assoc Applet");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEView", "Account Detail - Contacts View");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWECmd", "InvokeMethod");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWESP", "false");

// Original string was:
//   "1156341790828"
// Variablized string is:
//   "{$ DATETIME:MS1970 $}"
//
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWETS", TIME_SINCE_1970("milliseconds"));
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEIgnoreCtrlShift", "0");
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWEP", "1B_Account Contact List
Applet9_NewRecord0_0_1_7");

// Original string was:
//   "SWERowId0=1-7BZ"
// Variablized string is:
//   "{$ VAR:Siebel-7 $}"
//
//-- Update the Calculation Variable Siebel-7 --
//
// Variablized string is:
//   "SWERowId0={$ VAR: Siebel-6 $}"
//
__Siebel_7 = "SWERowId0=";
__Siebel_7 += __Siebel_6;
__Siebel_7 = __Siebel_7;
Set(NEXT_REQUEST_ONLY, POST_DATA, "SWERowIds", __Siebel_7);
Set(NEXT_REQUEST_ONLY, CHECKPOINT_NAME, "Page 20 - http://dtw-siebel78/callcenter");

Post_To("http://dtw-siebel78/callcenter_enu/start.swe");
DO_SLEEP(1);
```

Script Example with the Enable Siebel Support Option Not Selected

The following example has the Enable Siebel Support check box cleared.

```
Siebel_off
extern "C" int rhobot_script(PLAYER_INFO* s_info)
{
// Declare Variables
//



SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("cap_siebel_on.cpp");


SET_SCRIPT_LANGUAGE(SLID_English);
DO_InitHttp(s_info);
```

```
// Playback Options
//
Set(EVERY_REQUEST, PROXY_MODE, PROXY_AUTOMATIC_CONFIGURATION);
Set(EVERY_REQUEST, PROXY_SCRIPT, "http://proxyconf.compuware.com");
Set(EVERY_REQUEST, HTTP_VERSION, "1.1");
Set(EVERY_REQUEST, PROXY_HTTP_VERSION, "1.0");
Set(EVERY_REQUEST, JAVASCRIPT_LEVEL, FULL);
Set(EVERY_REQUEST, USER_PATIENCE, 120); // Maximum time to wait for an HTTP Reply
Set(EVERY_REQUEST, CONNECTION_RETRIES, 4);  // maximum attempts to connect
Set(EVERY_REQUEST, BROWSER_THREADS, 2); // total browser threads to simulate
Set(EVERY_REQUEST, CACHING, NO_CACHING);     // Enable/Disable cache
Set(EVERY_REQUEST, REUSE_CONNECTION, TRUE);   // maintain socket connection if possible
Set(EVERY_REQUEST, REUSE_SECURE_SESSION, TRUE);
..
..
..
Navigate_To("http://dtw-siebel78/callcenter_enu/start.swe");

//---------  REQUEST # 6  ---------
//
//  current page url is http://dtw-
siebel78/callcenter_enu/start.swe?SWECmd=GetCachedFrame&SWEACn=.....
//
Set(NEXT_REQUEST_ONLY, HEADER, "Accept", "image/gif, image/x-xbitmap, image/jpeg"
        ", image/pjpeg, application/x-shockwave-flash, */*");
Set(NEXT_REQUEST_ONLY, POST_DATA, "s_2_1_57_0", "Business");
Set(NEXT_REQUEST_ONLY, POST_DATA, "s_2_1_27_0", "sdfasdf");
Set(NEXT_REQUEST_ONLY, POST_DATA, "s_2_1_28_0", "sfdasfd");
}
```

Streaming Media

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, 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 is not requested until the media clip has finished being processed.

Recording Streaming Media

To record streaming media:

1. With a WWW session open in the Script Development Workbench, choose Options>Convert from the menu.
2. On the WWW tab, click Advanced.
3. Select the Streaming Media option and click OK.

When you record your transaction, make sure you invoke your media player through clicking a link or typing a URL in your web browser. QALoad records the URL from your web browser; therefore, you must use your browser to access your media file or else the URL will not be recorded in your script.

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 shows the title Real Media Request or Windows Media Request to indicate the type of request you recorded, and lists the following fields:

Requested URI: Lists the requested URI that invoked the media player. For Real Media the file typically is an RM file, while for Windows Media it is typically 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 Support

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.

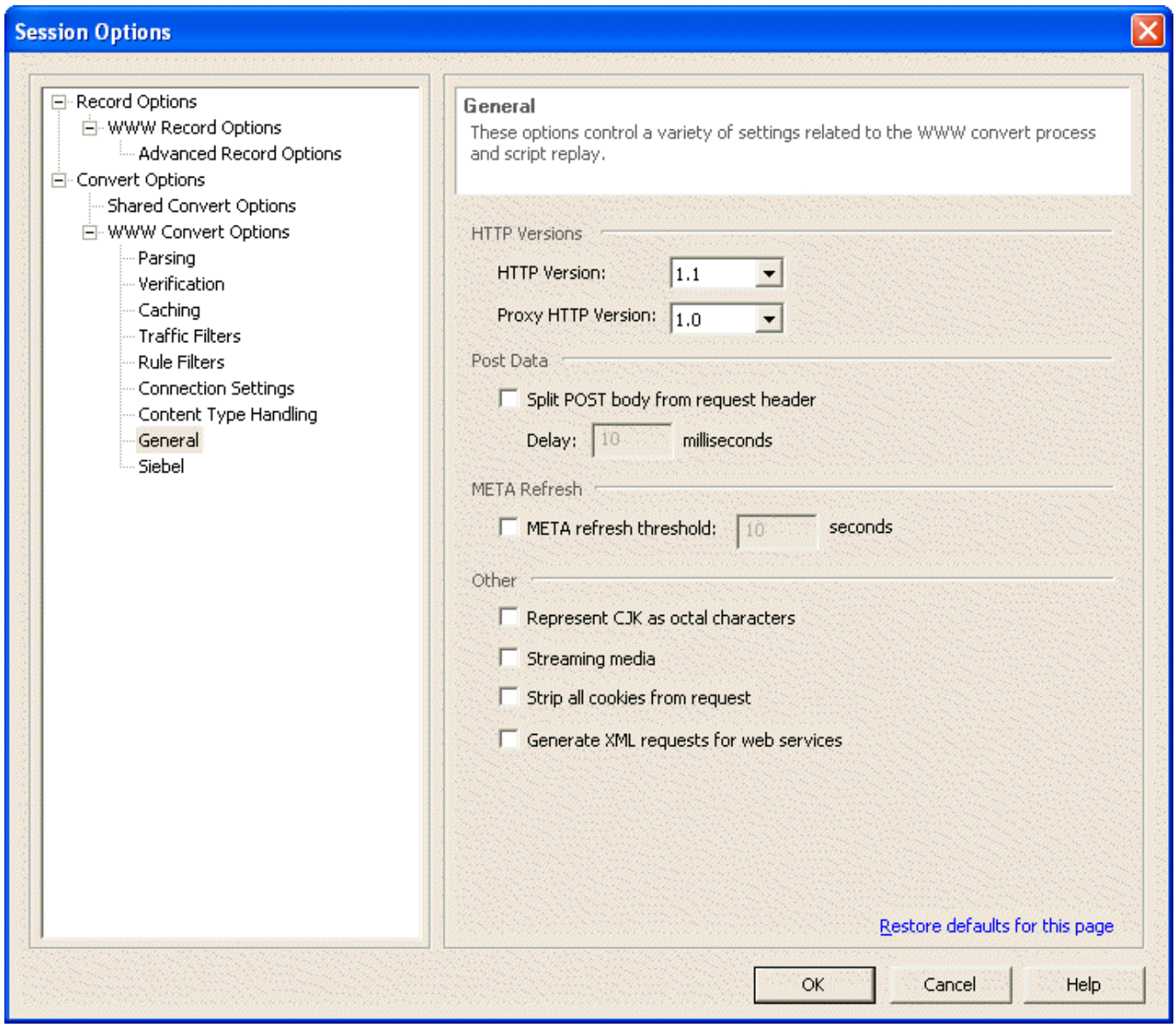
QALoad provides two methods of support for CJK: native character and encoding. Depending on your testing requirements and environment, it may be necessary to use both mechanisms to support the load testing of a Web site that contains CJK characters.

- 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.

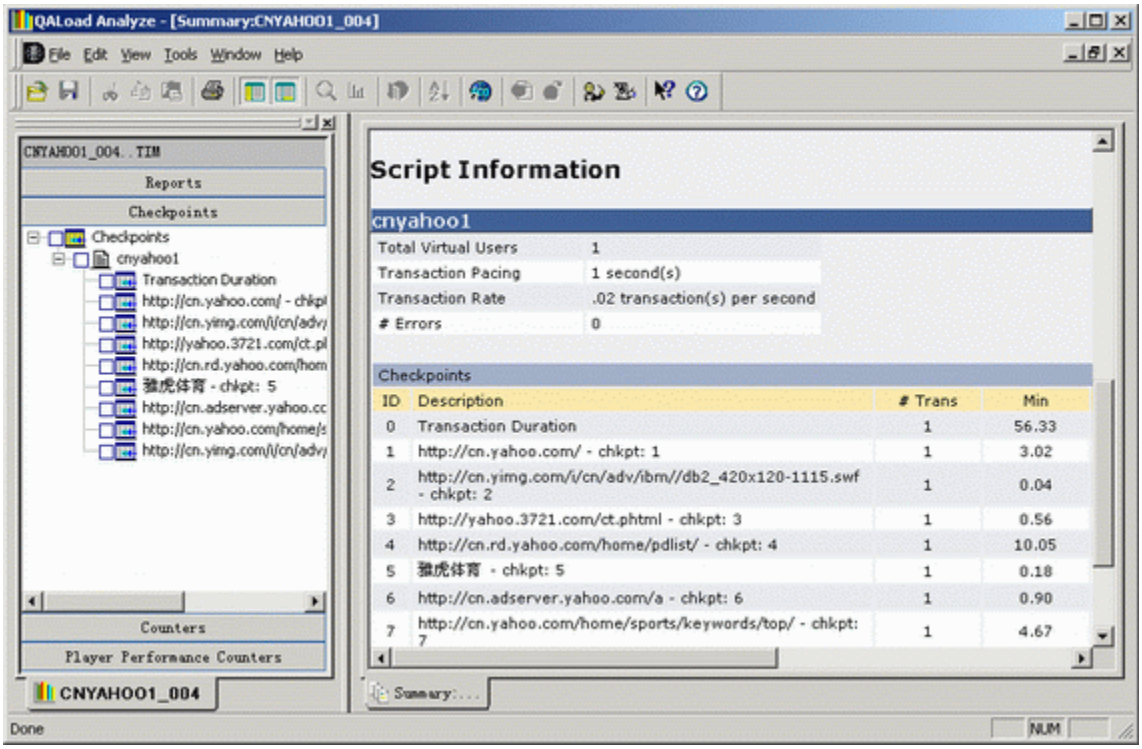
Native Character CJK Support

By default, QALoad translates Chinese, Japanese, and Korean (CJK) characters if you are using the QALoad Workbench on the same CJK language operating system as the operating system on which the Web application is being captured. There are no options to enable, but you may wish to ensure that the Represent CJK as Octal Characters option check box is not selected.

To verify, this option is located on the General page under WWW Convert Options in Options>Convert as shown below.



Below is an example of a timing file with native characters displayed in QALoad Analyze below.

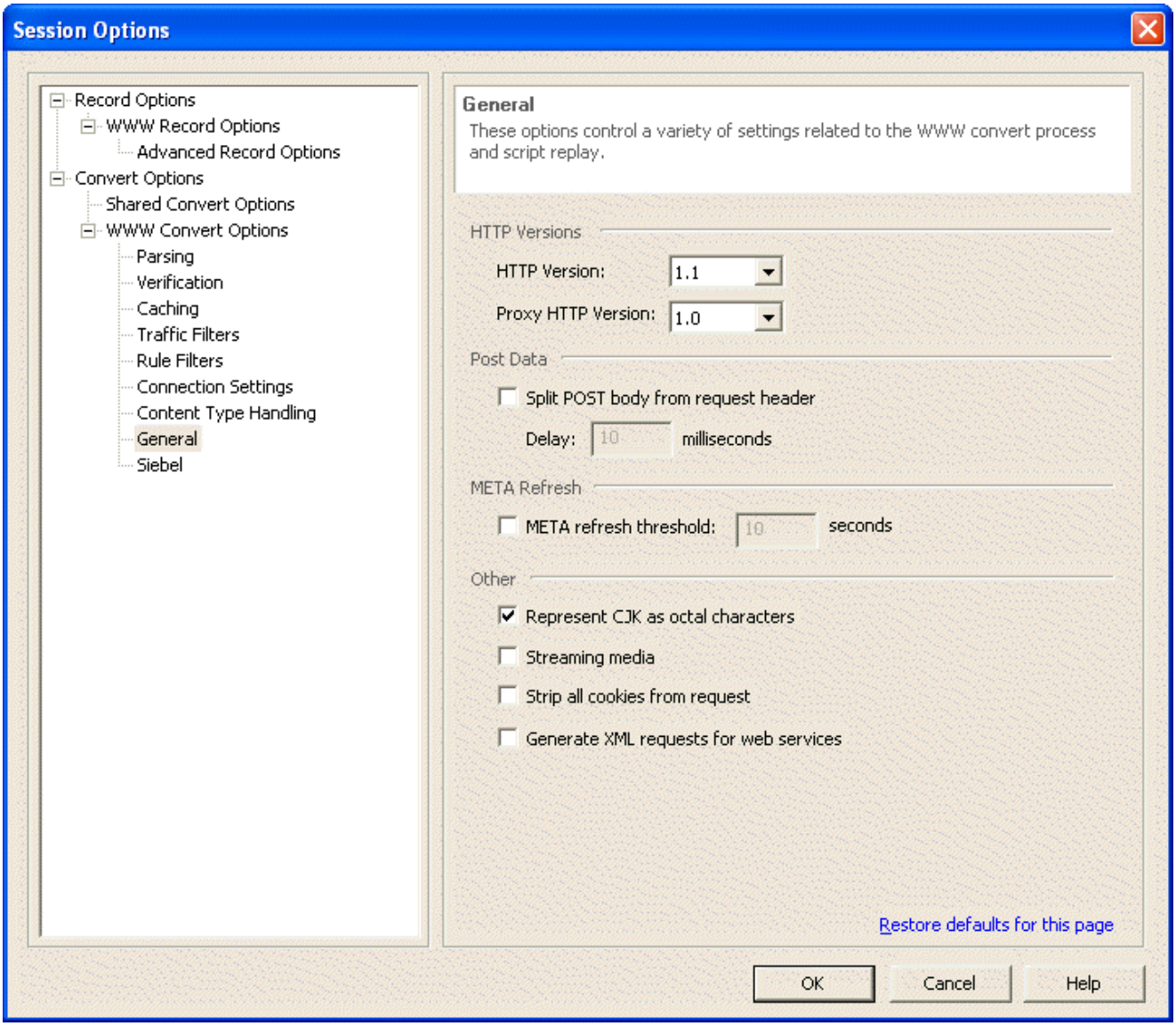


Encoding CJK Support

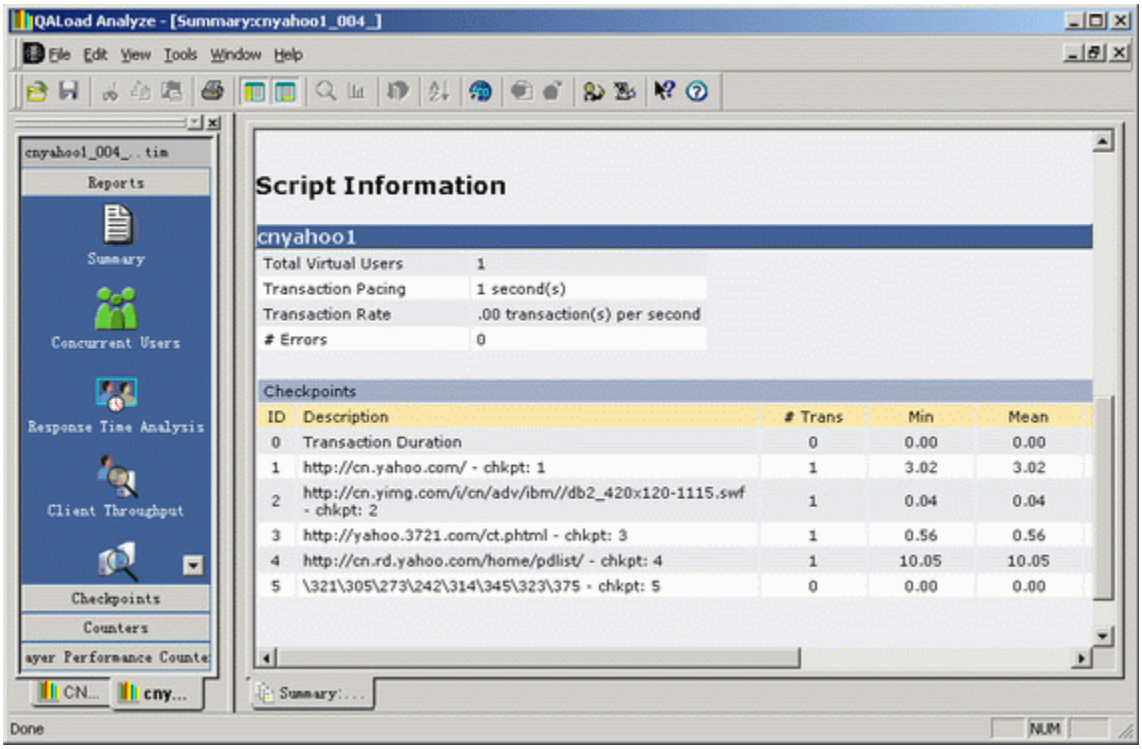
QALoad uses the encoding option when converting a capture file containing more than one set of Chinese, Japanese, or Korean (CJK) language characters. The encoding option also is used when converting the script on a CJK language operating system different than the one on which the application was captured.

To override QALoad's Native Character Support and enable Encoded Support:

1. In the QALoad Script Development Workbench, click Options>Convert. The Session Options dialog box appears.
2. In the tree view under WWW Convert Options, click General. The General convert options dialog box displays.
3. In the Other category, select the Represent CJK as Octal Characters check box as shown below.



When using encoded characters in QALoad test scripts, the resulting timing files do not display checkpoints in their native language. The following graphic of QALoad Analyze shows a timing file with encoded characters within it.



Represent CJK as Octal Characters

When this option is selected, the double-byte characters used for Chinese, Japanese, and Korean (CJK) scripts are converted into octal format.

Considerations

- 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.
- Data stays in encoded format throughout the load test: from capture, through convert and replay, to the analysis of the timing file.

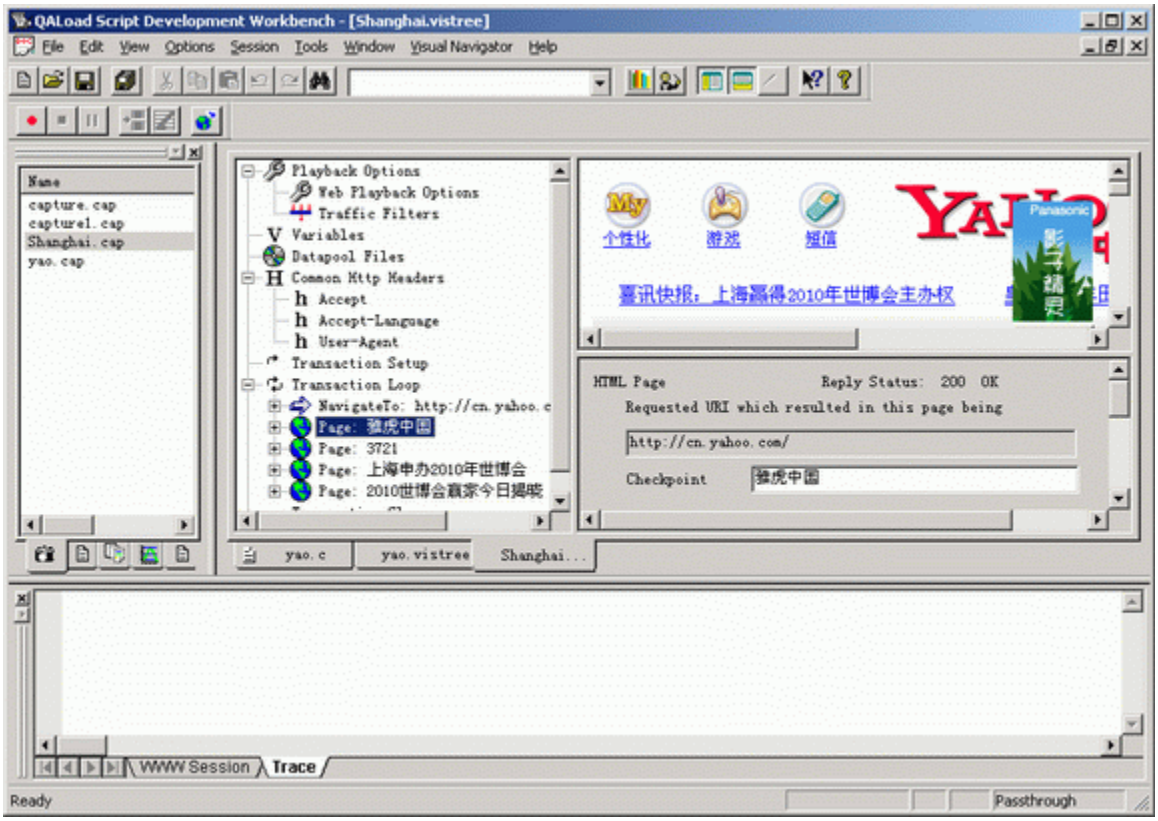
Use this option to:

- Enable the encoding of captured data from web applications containing Double Byte Character Sets (DBCS) such as Chinese, Japanese, or Korean.
- Encode all native characters when native character support cannot be used.

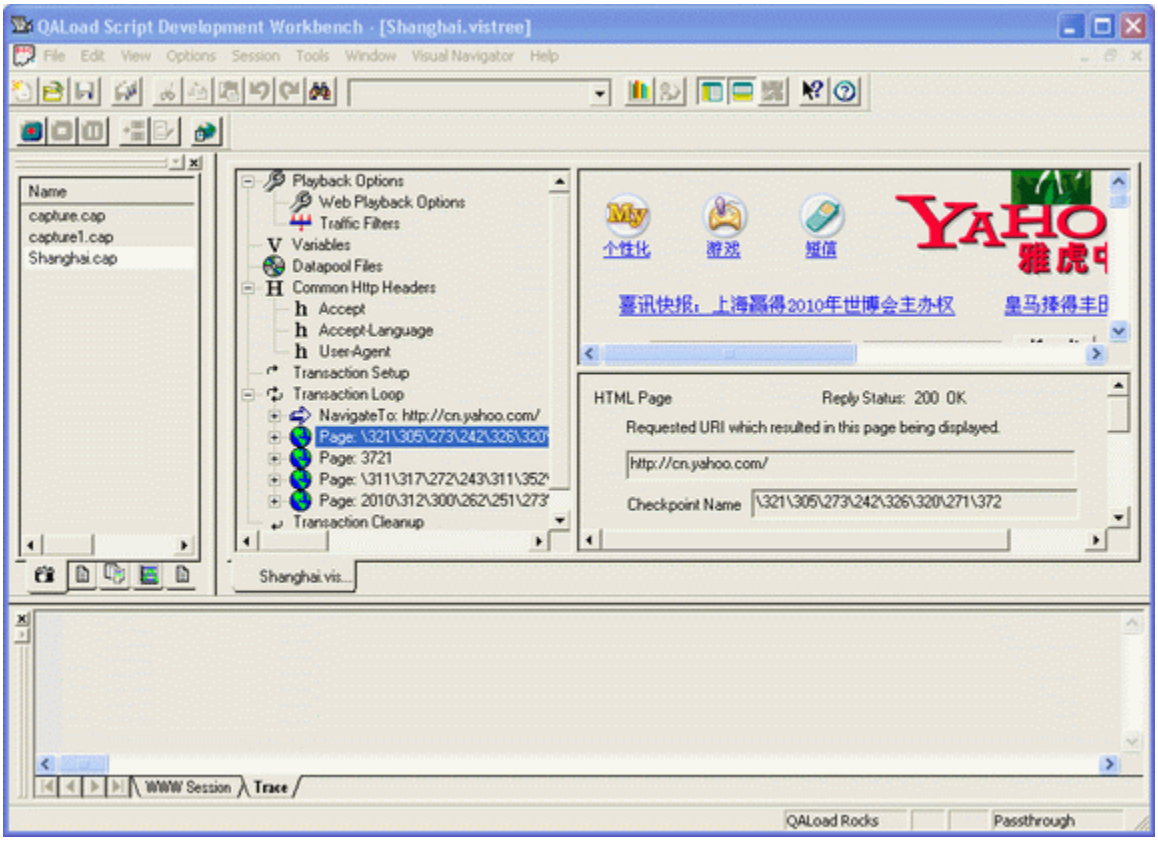
CJK and Visual Navigator

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.



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:

(WWW only) If you are running your application on Windows XP SP2, Compuware recommends you turn the 'pop-up blocker' feature off before recording a transaction. The feature can be disabled via the browser window's Tools menu in Internet Options>Privacy.

1. Open an appropriate middleware session in the QALoad 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. QALoad 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. When you have recorded a complete transaction, stop the application from which you are recording.
8. 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.

The Script Development Workbench automatically converts a capture file to a script file when you stop the recording process. 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.

Changing Recording Options without Recording

You can view and change recording options at any time without recording a capture file.

To change recording options without recording:

1. Access the QALoad Script Development Workbench. Details
2. From the Session menu, select the appropriate middleware or start a Universal session.
3. From the Options menu, choose Record. The Record Options wizard opens, showing a tab of recording options for the middleware you selected.
4. On your middleware tab, select options and enter information as appropriate.
5. Click OK.

For a description of available recording options, see Record Options Wizard.

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.

Recording with Manual Program Startup

Use this procedure to manually start your application and the Record facility. You can start your application at any time before or after starting to record.

This procedure only applies to Winsock and SAP.

To record with manual program startup:

1. Access the QALoad Script Development Workbench. Details
2. Choose the appropriate session from the Session menu.
3. From the Options menu, choose Record to open the Record Options wizard.
4. On your middleware tab, select the User Started Program option. Click OK.
5. WWW Only: Start your browser.

6. Click Start Record.
7. Start your application (SAP only: start QALSAP.EXE, which is located in the `\QALoad` directory).
8. Run the user operations you want to record. As you execute the operations, QALoad records the operations in a capture file (.cap).
9. (Optional) At any time during the recording process, you can insert any necessary commands or comments into the capture file. Details
10. When you finish, click Stop Record. You must stop recording before you shut down the application from which you are recording. QALoad prompts you to save your capture file. By default, capture files are saved in the `QALoad\Middlewares\<middleware_name>\Captures` directory.

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 QALoad 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 Session>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: (WWW Only) When you click Session>Convert, the WWW Script Conversion Mode Selection dialog box displays. Choose the mode in which to convert the script. How?
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?

Set Up Automatic Conversion and Compilation of a Script

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.

If the default settings to automatically convert a capture file have been changed, follow the steps below to reset the automatic conversion and compilation.

To set up automatic conversion and compilation:

1. From the Script Development Workbench menu, choose Options>Workbench.

2. On the Workbench Configuration tab, 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 is not overwritten accidentally.
6. Click OK to save your settings.

Editing a Script

Editing a Script

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 lists 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 the function 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 (one) 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 shows 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:

- Central: 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 QALoad Conductor. A central datapool is controlled by the QALoad Conductor, and you use the QALoad Conductor to set any options relating to a central datapool.
- Local: 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. Use the QALoad 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 QALoad 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. QALoad 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.

To assign a central datapool file:

1. With a session ID file open in the QALoad Conductor, click the Script Assignment tab.
2. In the External Data column for the selected script, click Browse.
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 QALoad 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 Add 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 QALoad 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.

To assign a local datapool:

1. Open a session in the QALoad 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 Add. 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 QALoad 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 QALoad 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 QALoad 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 QALoad 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.

     h. Assign a variable name for the selected string in the Variable Name field.

     i. 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. 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.

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. 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;

.
.
.
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();
```

> 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.

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 CtxSetWindowMatchTitle 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 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();
}
```

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 all Oracle E-Business Suite and Oracle Applications 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. QALoad automatically extracts ICX tickets and any necessary cookies from the WWW middleware traffic and passes them to the OFS middleware.

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 Record Options dialog box. Forms connection modes include server-side recording, HTTP, HTTPS, or socket.

Server-side recording is limited to applications that use Oracle Application Server. 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.

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 Oracle Forms Record 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, QALoad 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.

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, QALoad 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, QALoad wraps the binary data inside an HTTP stream and invokes Java's HTTP calls.
- For socket mode, QALoad 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.

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.

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

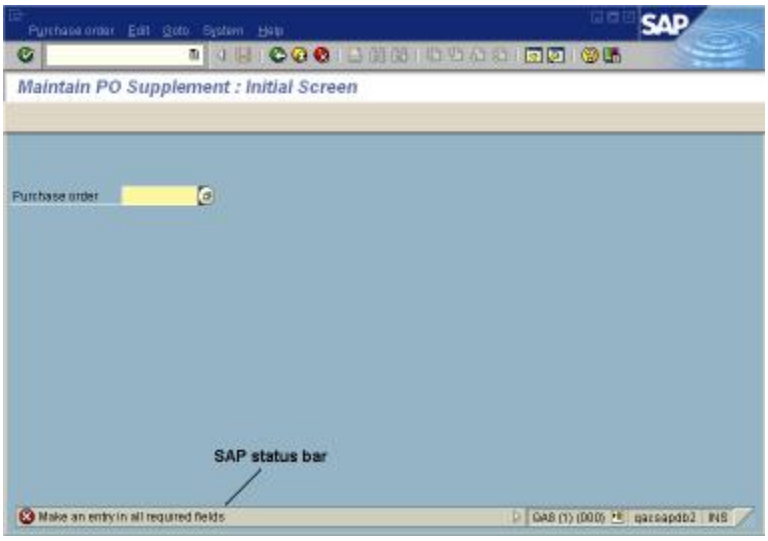
You may need to modify your script to handle multiple logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this 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:
   a. If the character is "\r", "\n", "\t", or "\f", it is represented in the script as a normal C escape character.
   b. 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.
   c. 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();
}
```

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 QALoad Conductor.
2. Click the Machine Assignment tab.
3. Select a script in the Script field.
4. Click Manager Players. The Manage Player Machines and Groups dialog box appears.
5. Double-click the Player machine name in the list. The Properties dialog box appears.
6. Select the Generate IP Spoof Data (machines with multiple IP addresses only) option in the Player Machine settings field.
7. 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 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 causes 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.

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();
```

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

CGI Requests

Simulating CGI Requests

The following topics describe strategies for simulating CGI requests:

CGI Parameter Encoding

CGI Get Requests

CGI Post Requests

CGI Forms

CGI Parameter Encoding

Common Gateway Interface (CGI) 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 using the DO_SetValue command.

When you modify parameters that are passed into a QALoad CGI request, ensure that all CGI parameters containing characters that are not alphanumeric (a-Z, 0-9) are encoded before 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 using the DO_EncodeString command.

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 Common Gateway Interface (CGI) program.
2. The server calls the CGI program, which usually returns a Web page. The returned page is called 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 the same way 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 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();
```

CGI Forms

Common Gateway Interface (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. The browser processes the action statement, gathers all input fields as name value pairs, and passes them to a CGI call contained in the action statement.

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();
```

Compiling a Script

Compiling a Test Script

A QALoad script is a real C++ 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 prompts you to compile the script.

Set Up Automatic Conversion and Compilation of a Script

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.

If the default settings to automatically convert a capture file have been changed, follow the steps below to reset the automatic conversion and compilation.

To set up automatic conversion and compilation:

1. From the Script Development Workbench menu, choose Options>Workbench.
2. On the Workbench Configuration tab, 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 is not overwritten accidentally.
6. Click OK to save your settings.

Testing a Script

Validating Scripts in Workbench

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.

Note: During validation of SAP scripts, do not minimize the SAP window. If the window is minimized, the validation may fail. This problem does not occur if you select the Hide Graphical User Interface for SAP Users option by clicking Browse [...] in the Type column of the Script Assignment tab in the Conductor. This SAPGUI option runs SAP on an alternate desktop that is not visible.

To configure the Script Development Workbench for validation:

1. Click Options>Workbench and select the Script Validation tab.
2. Select the Automatically Recompile check box if you want QALoad to compile a script before attempting to validate it. QALoad 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, if applicable.
5. Type a value in the Wait up to field. This is the number of seconds that the QALoad 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 QALoad 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 a script in Workbench:

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

Debugging a Script

If you encountered errors while validating or testing a script, use QALoad's debugging options to monitor the Player(s) that generated errors while they are running or after the test.

You can watch a virtual user execute a script on a Player Workstation while it is running. To monitor selected virtual users at runtime, enable the Debug Trace option before you run your test. Each virtual user for which you enabled Debug Trace displays messages on its assigned Player workstation indicating which commands are being executed.

You can instruct the Conductor to generate and save details about the script execution of selected virtual users by enabling Logfile Generation before you run your test. This applies to Citrix, ODBC, Oracle, Oracle Forms Server, SAP, Winsock, or WWW only.

To enable the debug options:

1. On the Conductor's Script Assignment tab, highlight the script you want to monitor.
2. In the Debug Options column, click the browse (...) button (note that the button may not be visible until you click in the Debug Options column).
3. On the Debug Options dialog box, you can optionally choose the following options:
   a. To enable the Debug Trace option: in the Debug Trace Virtual User Range area, choose which virtual users (if any) to monitor. You can choose None or All Virtual Users, or choose Virtual User(s) and then type the numbers assigned to the virtual users you want to monitor. You can monitor individual virtual users or ranges of virtual users.
   b. To enable Logfile Generation: in the Logfile Generation Virtual User Range area, choose which virtual users (if any) to monitor. You can choose None or All Virtual Users, or choose Virtual User(s) and then type the numbers assigned to the virtual users you want to monitor. You can monitor individual virtual users or ranges of virtual users.
4. Click OK to save your changes.
5. From the Conductor's main menu, click File>Save to save your test session ID.
6. Run your test as usual.

Note: Some log files are generated automatically when you run a test in the Script Development Workbench or Player.

QALoad Support Log Files

The table below identifies the QALoad log (support) files that are generated automatically during a test.

Note: Some log files are generated automatically when you run a test in the Script Development Workbench or Player.

Each virtual user for which you enabled Logfile Generation in Conductor creates a file containing information about their performance. After the test is finished, the Conductor requests all log files from the Players and save them in the directory `\Program Files\Compuware\QALoad\LogFiles` on the workstation where the Conductor is installed.

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

Middleware	.RIP File	.CAP File (Replay Capture)	.TRC File Player Trace	.LOG File
ADO	No	No	Yes	Yes
Citrix	Yes	Yes	Yes	No

ODBC	No	No	Yes	*Yes
Oracle	No	No	Yes	*Yes
Oracle Forms Server	Yes	Yes	Yes	No
SAP	No	No	Yes	Yes
Uniface	No	Yes	Yes	No
Winsock	No	No	Yes	*Yes
WWW	Yes	Yes	Yes	No

Note: *This middleware also uses the .log file in replay capture.

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.

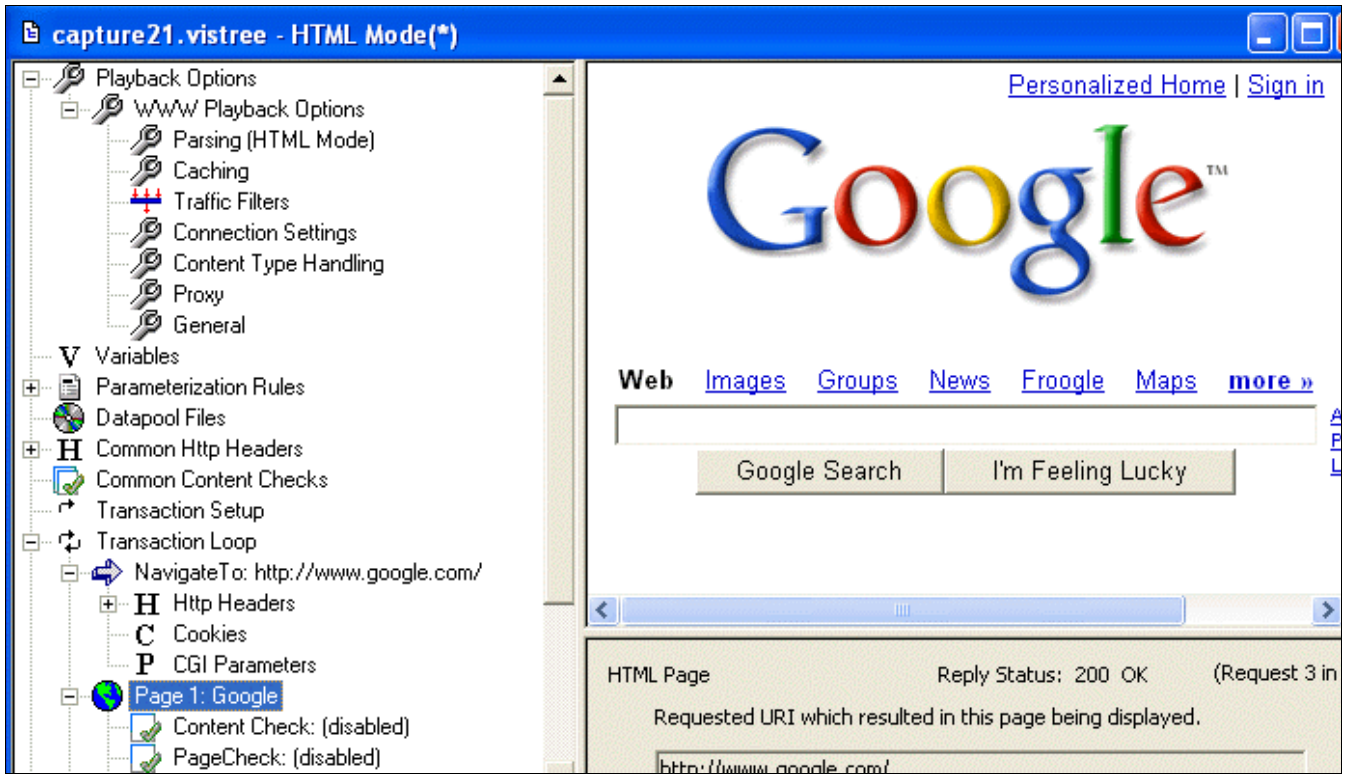
Visual Navigator (WWW)

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 Menus

The Visual Navigator has a number of special menu commands to help you develop your script.

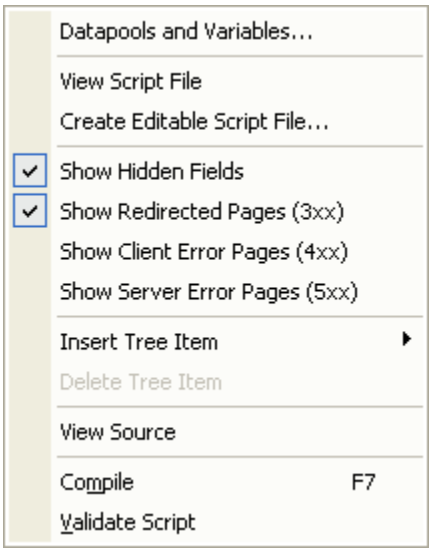
Visual Navigator Menu

Edit Menu

Tools Menu

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 right-click 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.

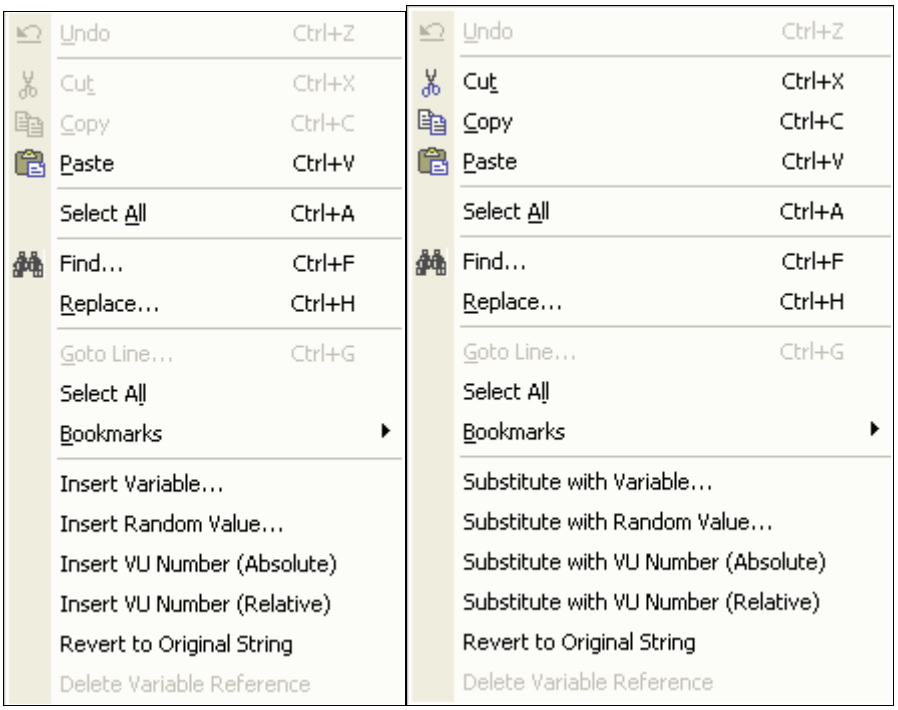
Delete Tree Item: Deletes the currently selected tree item. If the selected item may not be deleted from the script, this command is unavailable.

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 right-click 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 looks 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.

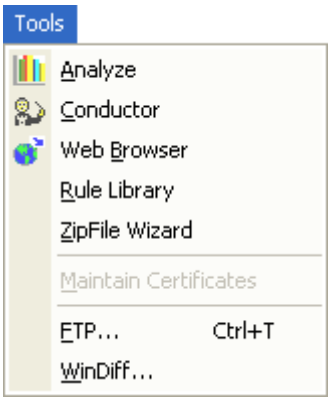
This menu item is enabled only if the edit box within the form can be variablized and it has been changed at some point.

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 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 $}.

To access Visual Navigator's Find feature:

1. Select Edit>Find in the toolbar. The Find and Replace dialog box displays with the Find tab on top.
2. Enter the appropriate options and click Find Next. When all matches are found, close the dialog box or click Replace to use the Replace feature.

To access Visual Navigator's Replace feature:

1. Select Edit>Replace in the toolbar. The Find and Replace dialog box displays with the Replace tab on top.
2. Enter the appropriate options.
3. Do one of the following:
    - Click Find Next to review each identified variable before replacing it.
    - Click Replace All to replace all variables without reviewing them.
4. Close the Find Replace dialog box.

Developing a Script Using the Visual Navigator (WWW)

Recording a Visual Navigator Script

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 records all sent and received HTTP and SSL calls using the Script Development Workbench's Web proxy and writes the activity to a capture file.

After recording, the capture file must be converted to an editable, C++ 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:

If you are running your application on Windows XP SP2, Compuware recommends you turn the 'pop-up blocker' feature off before recording a transaction. The feature can be disabled via the browser window's Tools menu in Internet Options>Privacy.

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. Click Options>Convert. The Session Options dialog box opens. Set any applicable options on the Shared Convert Options and WWW Convert Options screens. Click OK. (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.)
5. From the toolbar, click the Start Record button. QALoad 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 are prompted to save your capture file. By default, capture files (.cap) are saved in the directory `QALoad\Middlewares\WWW\captures`. In the File Name field, type a file name and click OK. The WWW Script Conversion Mode Selection dialog box appears.

   Note: If you previously set options to prompt automatic conversion, your capture file is converted to a Visual Script and opens automatically in the editor. For more information about setting up automatic conversion, see Configuring the Script Development Workbench.
8. Select either HTML Mode or HTTP Mode. Click OK.
9. Click the Captures tab and locate the capture file you just saved.
10. Right-click the file and choose Convert. QALoad converts your capture file to a Visual Navigator script and opens 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.

Note: If you choose to enable support for Siebel while converting a capture file, you must copy the Siebel files from the Siebel installation folder to `C:\Program Files\Compuware\QALoad\BinaryFiles`.

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/Decrement/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.

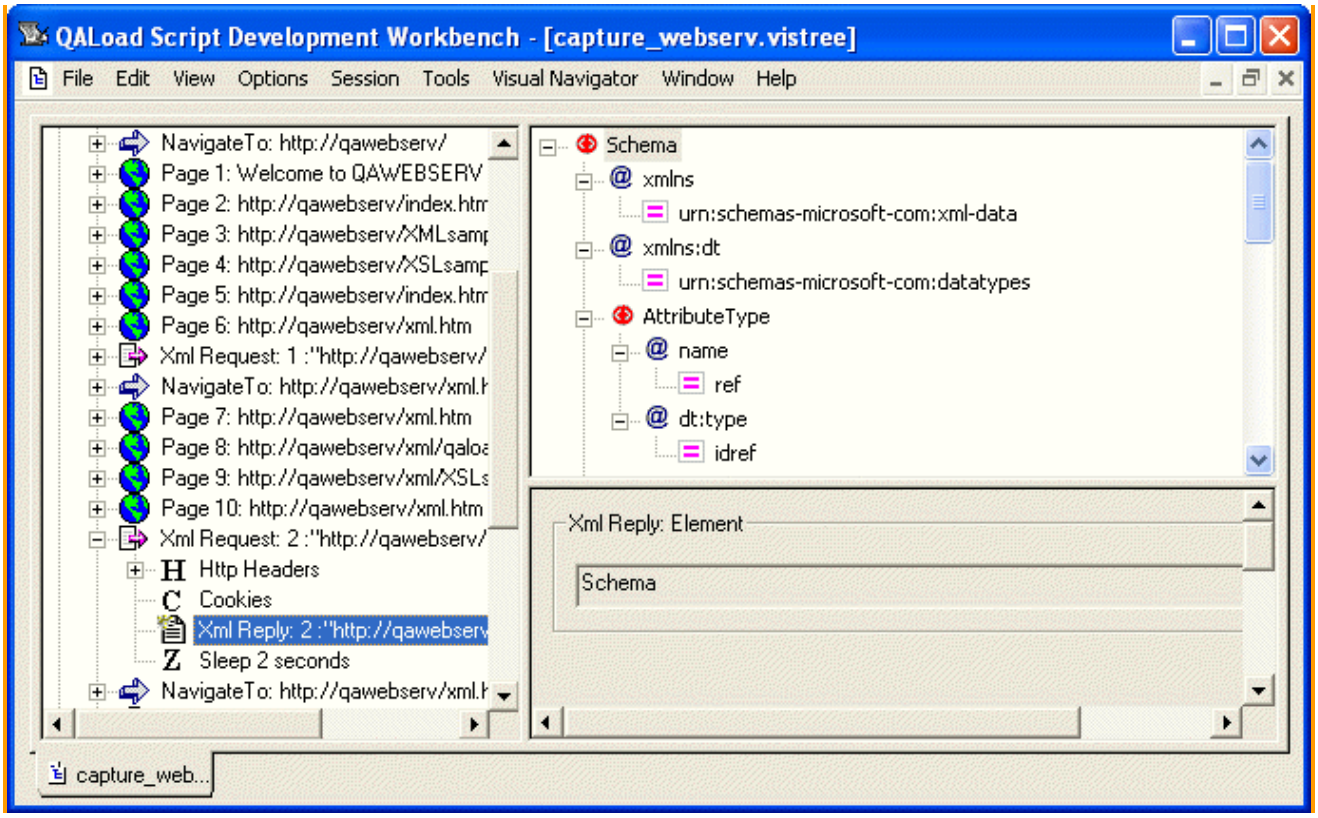
XML Support

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, Compuware recommends 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, the Script Development Workbench toolbars and panes that are not directly related to this help topic are not displayed. 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. The status code is 200 OK for most pages that return correctly.

Request URI: This read-only field shows the URI requested, which resulted in this page being displayed.

Checkpoint Name: If the page has a title, it is used as the checkpoint name. If not, the word Checkpoint is used. To make sure all checkpoint names are unique, QALoad adds 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, it is sent for that request only. If the header has the same name as one of the common headers, it overrides 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 contains 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.

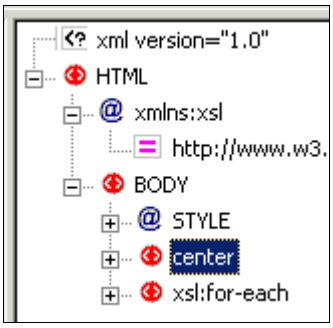
What if no XML data is associated with a request?

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 displays 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 displays 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 is 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 is 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 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.

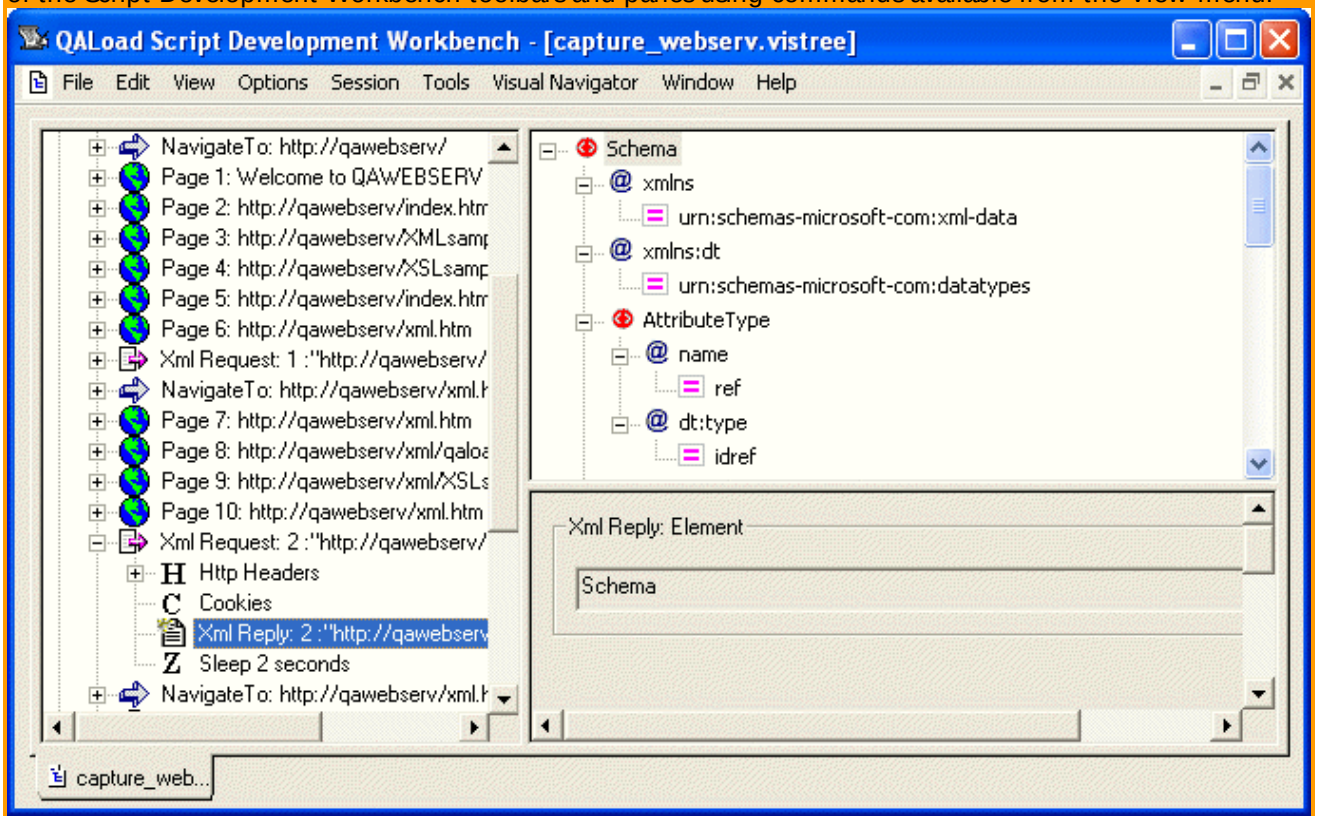
XML Request: Requests for XML documents are denoted by a document/arrow icon underneath the Transaction Loop element. These represent the requests for XML data made during the transaction that was recorded. XML Request items can be parent to a number of lower-level script sub-items in the tree-view, 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, Compuware recommends 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, the Script Development Workbench toolbars and panes that are not directly related to this help topic are not displayed. 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 shows the title Real Media Request or Windows Media Request to indicate the type of request you recorded, and lists the following fields:

Requested URI: Lists the requested URI that invoked the media player. For Real Media the file typically is an RM file, while for Windows Media it is typically 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.

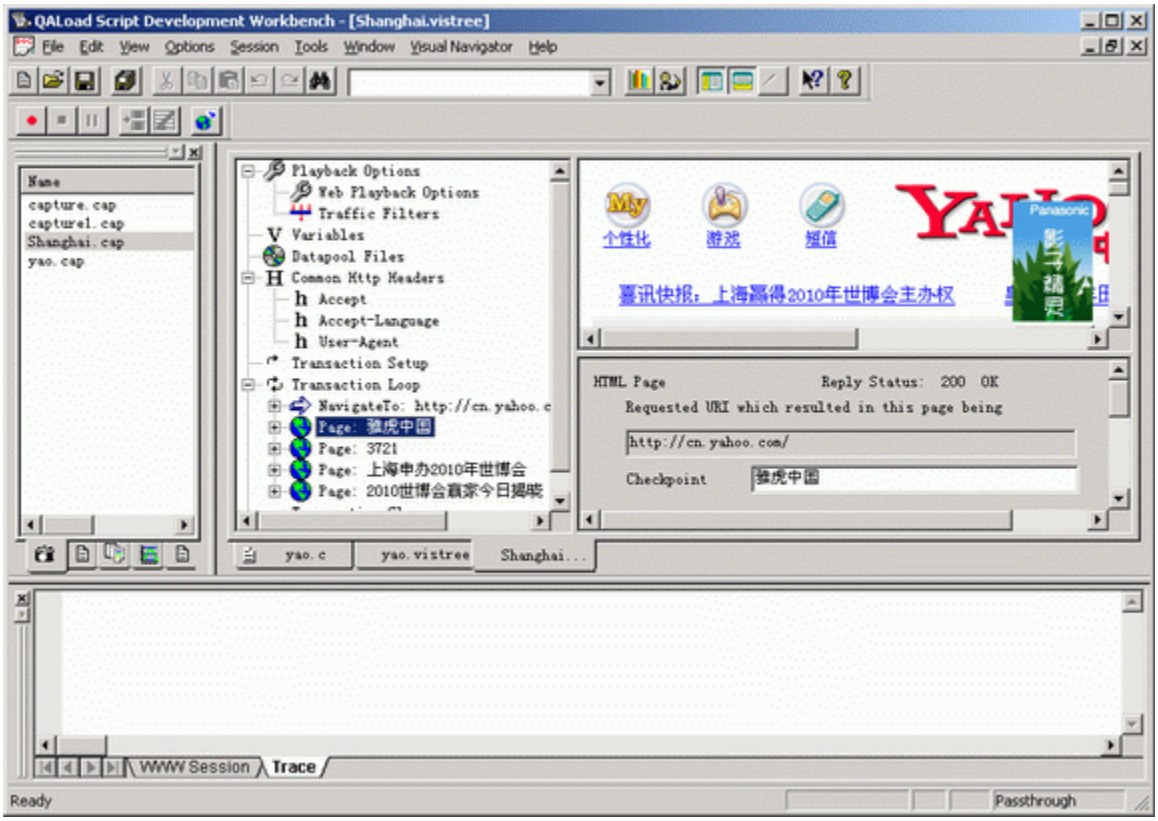
Play Requested Media for N seconds: You can specify how much of the streaming media file the virtual user should play, in seconds, before moving on to the next request. A value of zero indicates that the entire media stream should be played.

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.

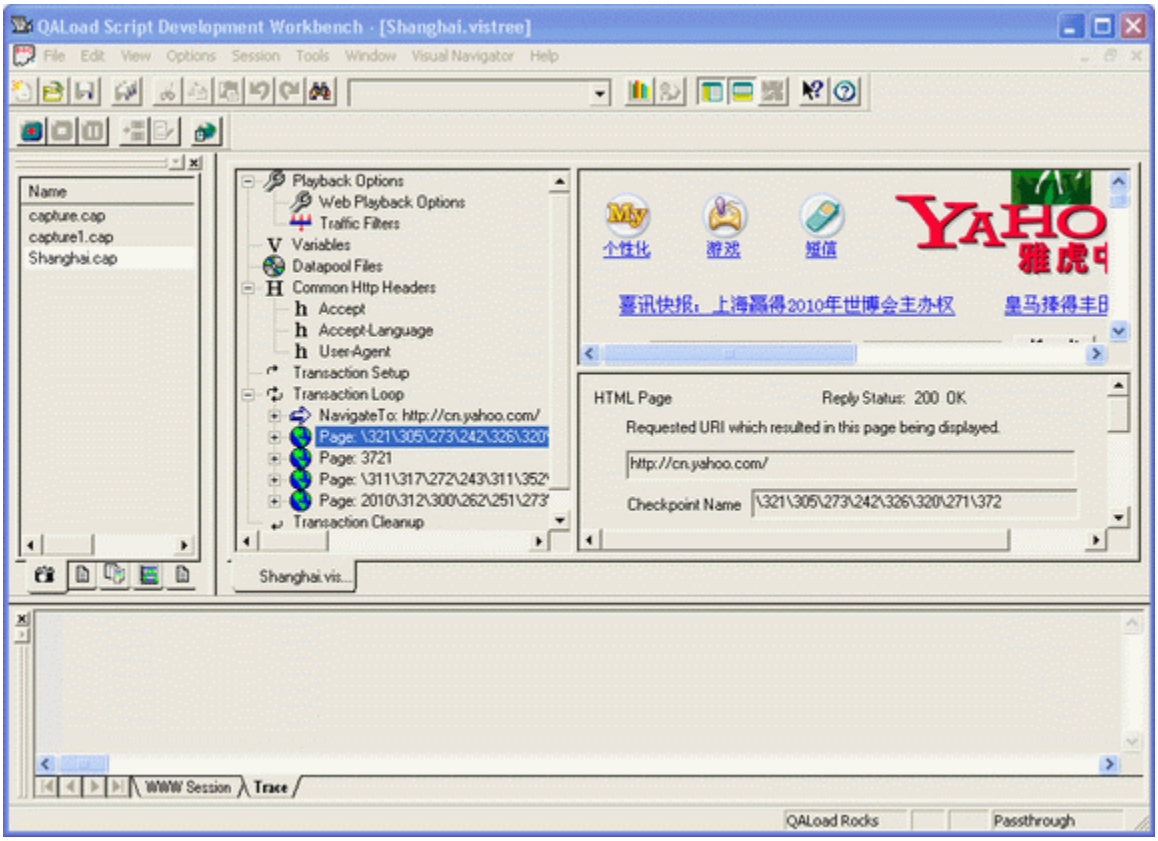
CJK and Visual Navigator

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

Primary 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:

WWW Playback Options

Variables

Parameterization Rules

Datapool Files

Common Http Headers

Common Content Checks

Transaction Setup

Transaction Loop

Transaction Cleanup

WWW 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.

Parsing

Caching

Traffic Filters

Connection Settings

Content Type Handling

Proxy

General

Siebel

Variables

These are local variables that have been created for this script. When you highlight a variable in the tree view, the fields display data related to the variable type you selected. For information about parameterization, see Parameterization.

Parameterization Rules Script Items

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.

Note: Properties for the rules are defined using the Rule Library Wizard or the Variable Replacement Wizard.

Datapool Files

The 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.

Move UP/Move DOWN: Click the arrow to move the selected transaction loop item in the tree-view up or down.

The following items are automatically cleared at the end of a transaction. When you move the transaction loop, you can prevent memory leaks by selecting the items to clear at the end of the transaction.

Cookies: Select this checkbox to clear all cookies at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(ALL_COOKIES) statement in the generated script.

Cache: Select this checkbox to clear the WWW cache at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(CACHE) statement in the generated script.

Connections: Select this checkbox to close any remaining connections at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(CONNECTIONS) statement in the generated script.

Referring Page: Select this checkbox to clear the HTML referring page at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(REFERER) statement in the generated script.

Basic Authorization: Select this checkbox to clear the basic NTLM authorization at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(BASIC_AUTH_FLAG) statement in the generated script.

Proxy Authorization: Select this checkbox to clear the proxy authorization at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(PROXY_AUTH_FLAG) statement in the generated script.

Transaction Cleanup

Lists actions that occur after the script has finished executing the appropriate number of transactions. Any items that occur under this heading are 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.

Move UP/Move DOWN: Click the arrow to move the selected transaction cleanup item in the tree-view up or down.

The following items are automatically cleared at the end of a transaction. When you move the transaction cleanup item, you can prevent memory leaks by selecting the items to clear at the end of the transaction.

Cookies: Select this checkbox to clear all cookies at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(ALL_COOKIES) statement in the generated script.

Cache: Select this checkbox to clear the WWW cache at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(CACHE) statement in the generated script.

Connections: Select this checkbox to close any remaining connections at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(CONNECTIONS) statement in the generated script.

Referring Page: Select this checkbox to clear the HTML referring page at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(REFERER) statement in the generated script.

Basic Authorization: Select this checkbox to clear the basic NTLM authorization at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(BASIC_AUTH_FLAG) statement in the generated script.

Proxy Authorization: Select this checkbox to clear the proxy authorization at the end of the transaction. This checkbox outputs the appropriate statement for the Clear(PROXY_AUTH_FLAG) statement in the generated script.

Transaction Loop Items

Transaction Loop Items

The following items can exist under a Transaction Loop item in the tree-view:

- Synch
- IP Spoof
- Read Datapool
- Checkpoint pair
- Increment/Decrement/Reset Variable
- Debug Print
- Comment

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 IPSPOOF.dat in the Visual Navigator tree-view. For more information about creating this datapool file and inserting it, see Setting Up IP Spoofing.

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.

Checkpoint Pair Script Item

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.

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.

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 META Refresh option was selected on the WWW Convert Options General 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.

PageCheck Sub-item

Page Check enables you to verify that the title of the page that was requested is correct.

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.

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.

Sleep Sub-item

Every page has 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).

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.

Frames

When an HTML page that is recorded contains frames, they are represented in the tree-view (left pane) with a circle icon containing a capital F. A frame page is indented beneath the page that is its parent. If you click on a frame icon in the tree-view, the corresponding frame is 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.

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.

Click On Link Sub-item

When the user clicks on a text link or an image link, then a Click On Link action item is 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) is 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 determines which region was clicked on and requests the page linked to 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 sends the server the mouse coordinate relative to the top-left corner of the image. The server then replies with the appropriate page.

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.

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.

Action Item Sub-Items

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

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.

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 displays 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 is generated in the script.

You can insert additional cookies into the Cookies section of a page as another means of variablizing the playback. How?

CGI Parameters Sub-item

Lists CGI parameters sent along with the request made by the Action item.

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.

Forms

Forms Sub-items

Many pages that are used during a WWW load testing session contain forms that a user must fill out, and submit buttons that are 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). 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).

Underneath the Fill In Form item in the tree view are Form Element items, such as Edit Boxes, Radio Buttons, and Check Boxes, representing elements that can appear in forms on HTML pages. Following the Fill In Form item is either a Click On Button item or a PostTo item.

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: Name of the Checkbox.
Value: 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.

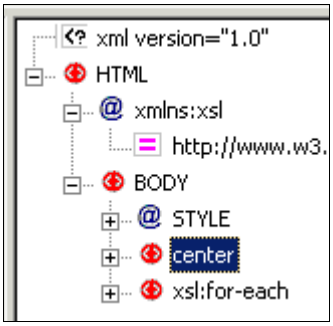
What if no XML data is associated with a request?

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 displays 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 displays 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 is 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 is 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

Parameterization in the Visual Navigator

Parameterization

Parameterization is the process of substituting certain values in a script with variables you define. Parameterization is used in Visual Scripting for WWW scripts.

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.

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

Overview of Variables

When you record a transaction, the resulting script is a recording of the actions of a single end-user. When you play back that script multiple times during a load test, you probably want it to emulate the actions of multiple users making differing requests of your server instead of the single user that was recorded. One way to achieve that is to replace certain data with a variable that draws its value from a list of values that you provide. Here are some examples of why and how you might use variables in a script:

- If your original script recorded a user logging on to a site using an ID and password, you can replace the ID and password with variables in the script. At test time, those variables draw their values from a datapool file of acceptable values, using a different set of values for each transaction run. In other words, that one script could emulate a number of different users by utilizing a different user ID/password combination for each transaction.
- If your script inserted new records into a customer database, you might want the names to be unique each time the script is run (each transaction). You could create a datapool file of names, and then insert a variable into the script where the name was typed. At test time, the variable inserts a different name from the datapool file with each transaction.
- If an ID string is returned from the server and that ID is then used as part of future requests to the server, and each virtual user may get back a different ID from the server, you can use a variable to

use a specific ID. You could extract an ID from the reply, place it into a variable, and then use the value in that variable in place of the actual ID for future requests, ensuring you only use the ID you specify.

There are a number of values in your script that can be replaced with variables. Those values are noted in the form view (bottom pane) with var.... Values that use the Variable Replacement Wizard are shown in the form view with Var Wiz. Typically variables derive their values from a datapool an extracted string, or a calculated value.

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 Rename.

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 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 and find this copy of the extracted text. You can increase or decrease the size of the 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.

Editing Variables

Adding a Variable

Add variables to the 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.

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:

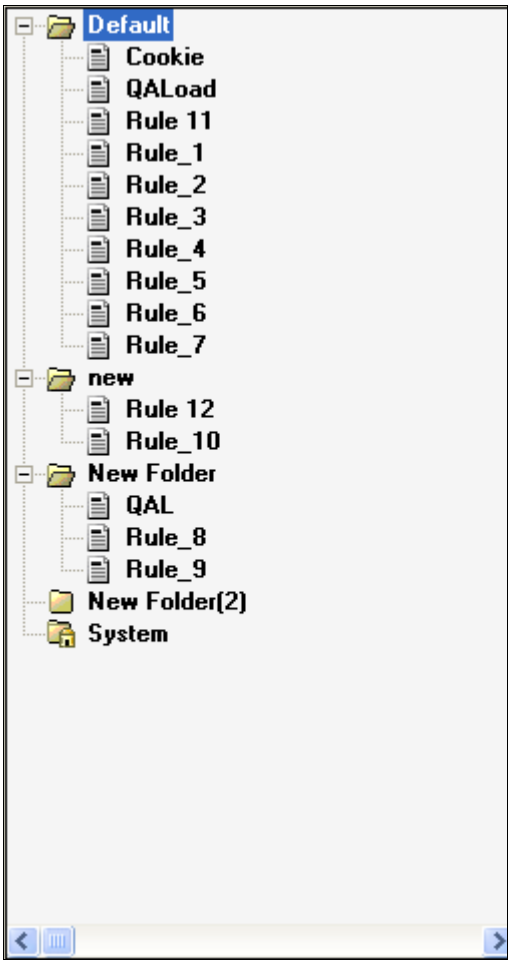
- Menus
- Tree view
- Details

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.

Applying a Rule

When you define values to replace variables in a script, you can save them as rules and apply them to other scripts.

To apply a rule from the Visual Tree:

1. In the Visual Navigator vistree, click the Parameterization Rules script item and select the rule 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 each matching item 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, a 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.

Sample Scripts

Overview — Sample Scripts

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.

Tips: Uppercase characters are not captured when the CAPS Lock key is on. Manually modify the script to use uppercase characters or hold down the SHIFT key during recording.
Similarly, the Windows Logo key is not supported. Do not use the Windows Logo key to start applications while recording a Citrix script.

The scripts listed below include a description of the problem, the procedure for implementing the modification, and samples of a modified script.

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

Flushing Past Events from the Internal Queue

Handling Citrix Server Farms

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. 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.

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. 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;

.
.
.

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();
```

> 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.

Handling Dynamic Window Titles

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"

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"

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 CtxWaitForWindowCreate 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 CtxWaitForWindowCreate 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 scripts illustrate this process.

Sample original script

Sample temporary script

Sample modified script

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 following sample scripts illustrate this process:

Sample original script

Sample modified script

Scripting Mouse Actions

Modifying the Script for Controlling Mouse Actions

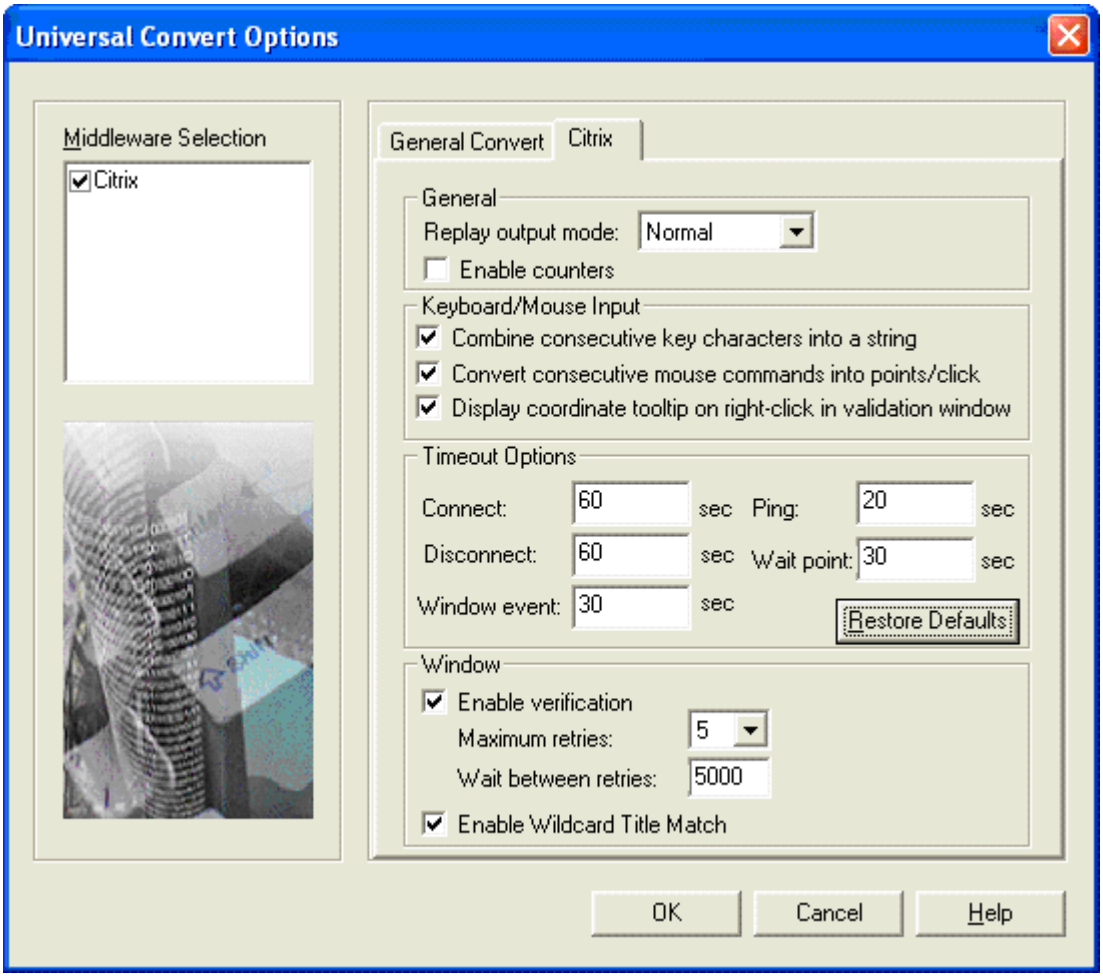
Changes to screen resolution or server settings can cause the appearance, size, and location coordinates of a window to differ from the expected behavior in the recorded script. By setting Display coordinate tooltip on right-click option in validation window in the Convert Options dialog box, you can 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.

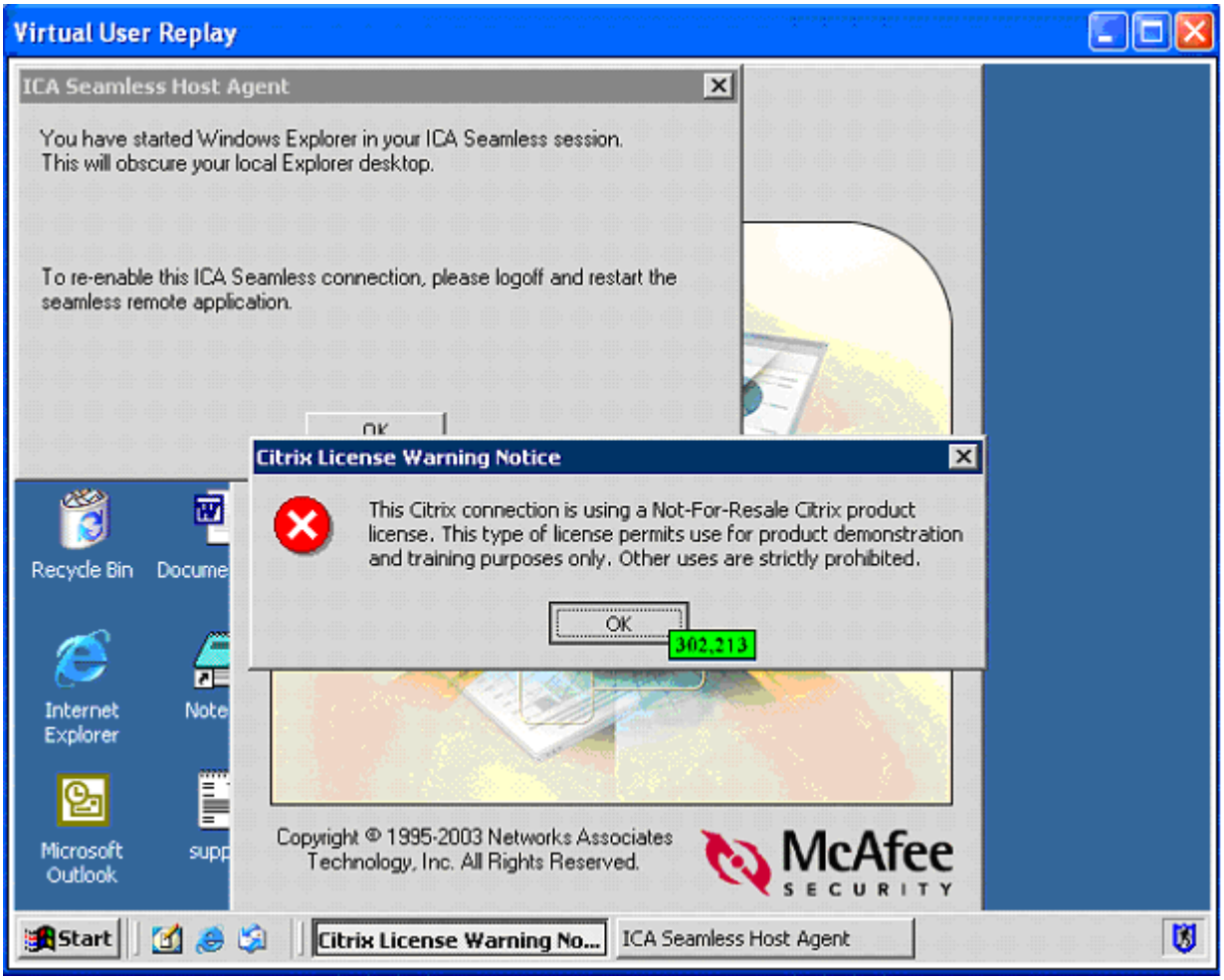
To validate the settings:

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 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…

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

Changing Accessibility Options in Oracle EBS-12

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

Checking the SAP Status Bar

Extracting a String from a SAP Control

Extracting a Unique String from a SAP Control

Handling Multiple SAP Logons in a Single Script

Implementing Content Check of a SAP Control

Required SAP Commands to Support Transaction Restart

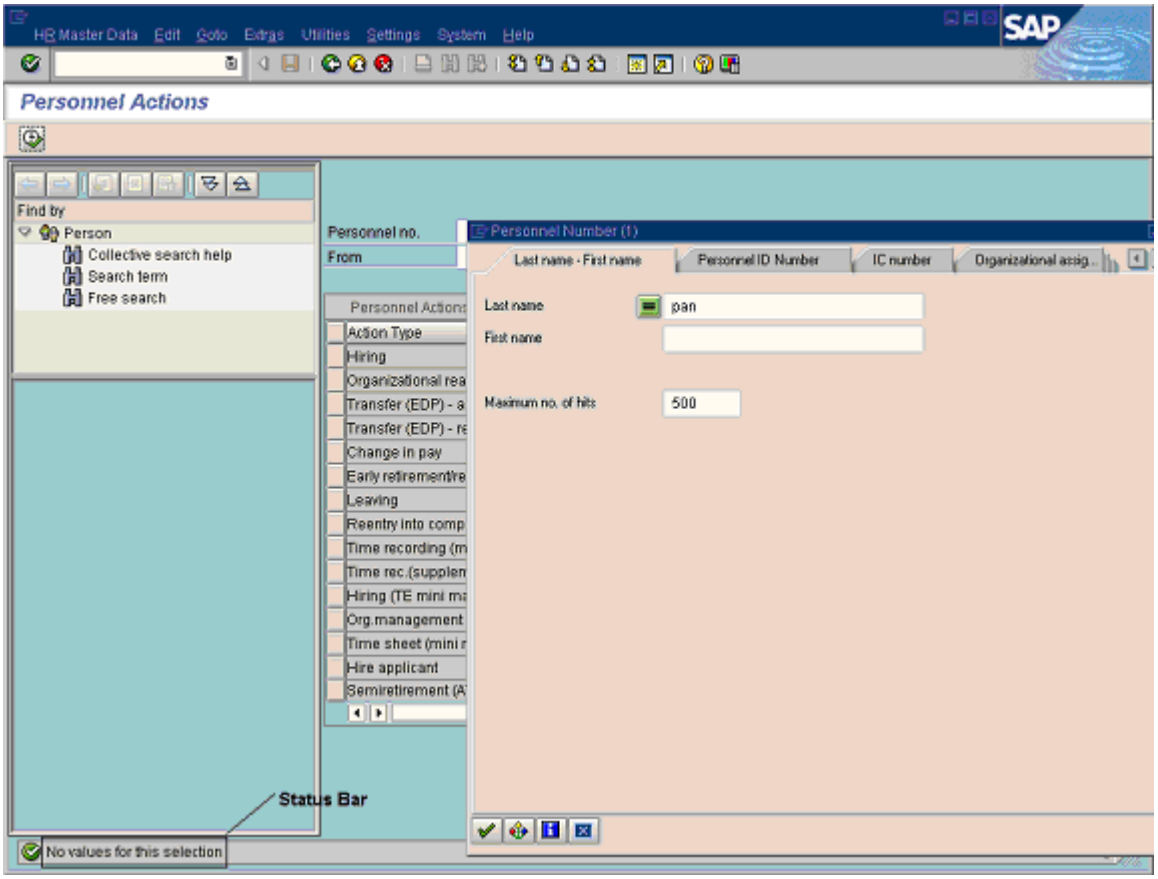
Required SAP Script Commands

Retrieving SAP Counter Data

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.

See Also

SAP Scripts Overview

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);
```

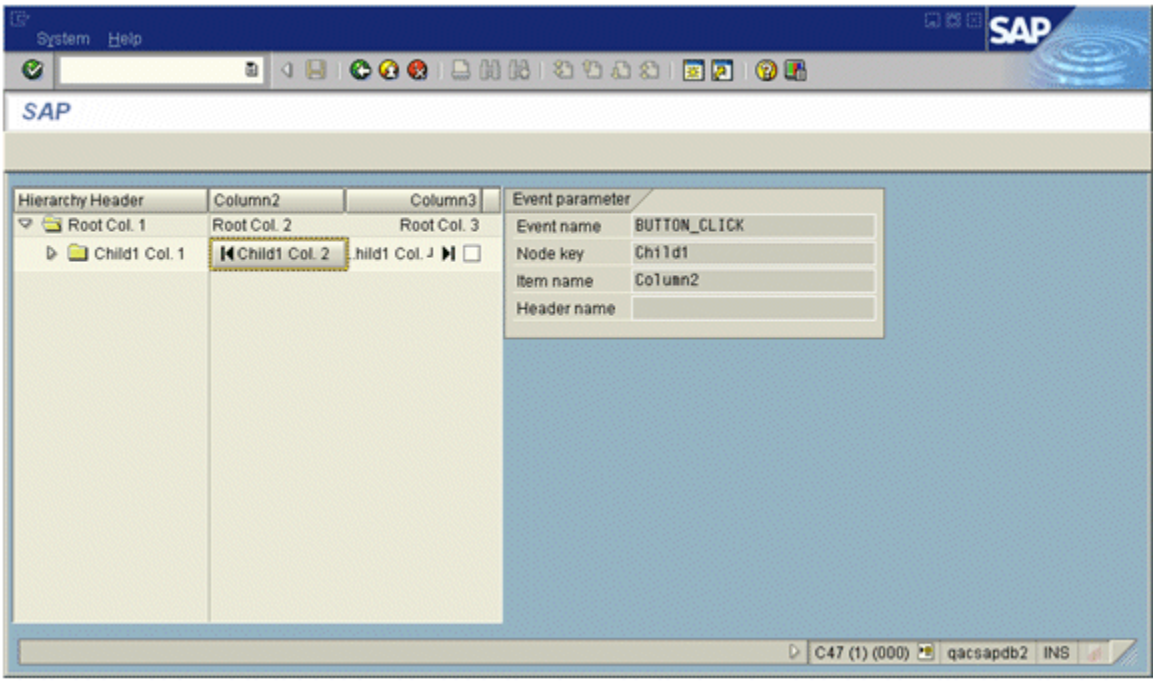
```
SAPGuiPropIdStr(""
   "wnd[1]/usr/tabsG_SELONETABSTRIP/tabpTAB001/ssubSUBSCR_PRESEL:SAPLSDH4:0220/sub:"
"SAPLSDH4:0220/txtG_SELFLD_TAB-LOW[0,24]" ); //1109615036.231
SAPGuiCmd1(GuiTextField,PutText,"pan");
SAPGuiCmd1(GuiTextField,PutCaretPosition,3);

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");
}
...
```

Extracting a String from a SAP Control

Overview: Extracting a String from a SAP Control

When running an SAP script, you may want to get the control's text.

For example, in the following screen, after Child1 Col. 2 is selected, the right table is filled in with data by the SAP server. You can use the information as needed.



Use the SAPGuiGetControlText command to extract data from a SAP server returned control. The sample script illustrates this procedure.

Sample: Extracting a String from a SAP Control

The following sample script shows the SAP commands required to extract a data string from a returned control in an SAP environment. You can use SAPGuiGetControlText to get the content of the Event name

textfield. Use convert option Insert SAP control comments: all controls to output all controls as comment.

Sample Script

```
     SAPGuiPropIdStr("wnd[0]/usr/cntlTREE_CONTAINER/shellcont/shell"
                  );//1152817068.134
     SAPGuiCmd2(GuiCtrlTree,SelectItem,"Child1","Column2");
     SAPGuiCmd2(GuiCtrlTree,EnsureVisibleHorizontalItem,"Child1","Column2");
     SAPGuiCmd2(GuiCtrlTree,PressButton,"Child1","Column2");
     SAPGuiCheckScreen("DWDM","SAPCOLUMN_TREE_CONTROL_DEMO","SAP");//1152817068.321

           // *SAP* GuiMenubar   name="mbar"   Id="wnd[0]/mbar"
           // *SAP* GuiMenu   name="System"   Id="wnd[0]/mbar/menu[0]"
           // *SAP* GuiMenu   name="Create Session"   Id="wnd[0]/mbar/menu[0]/menu[0]"
           // *SAP* GuiMenu   name="End Session"   Id="wnd[0]/mbar/menu[0]/menu[1]"
           // *SAP* GuiMenu   name="User Profile"   Id="wnd[0]/mbar/menu[0]/menu[2]"
                     …..
                    // *SAP* GuiLabel   name="%#AUTOTEXT004"
Id="wnd[0]/usr/lbl%#AUTOTEXT004"
           // *SAP* GuiTextField   name="G_EVENT"   Id="wnd[0]/usr/txtG_EVENT"
           // *SAP* GuiLabel   name="%#AUTOTEXT002"   Id="wnd[0]/usr/lbl%#AUTOTEXT002"
           // *SAP* GuiTextField   name="G_NODE_KEY"   Id="wnd[0]/usr/txtG_NODE_KEY"
           // *SAP* GuiLabel   name="%#AUTOTEXT005"   Id="wnd[0]/usr/lbl%#AUTOTEXT005"
     // *SAP* GuiTextField   name="G_ITEM_NAME"   Id="wnd[0]/usr/txtG_ITEM_NAME"
     // *SAP* GuiLabel   name="%#AUTOTEXT006"   Id="wnd[0]/usr/lbl%#AUTOTEXT006"
     // *SAP* GuiTextField   name="G_HEADER_NAME"   Id="wnd[0]/usr/txtG_HEADER_NAME"
     // *SAP* GuiStatusbar   name="sbar"   Id="wnd[0]/sbar"
     // *SAP* GuiStatusPane   name="pane[0]"   Id="wnd[0]/sbar/pane[0]"
     // *SAP* GuiStatusPane   name="pane[1]"   Id="wnd[0]/sbar/pane[1]"
     // *SAP* GuiStatusPane   name="pane[2]"   Id="wnd[0]/sbar/pane[2]"
     // *SAP* GuiStatusPane   name="pane[3]"   Id="wnd[0]/sbar/pane[3]"
     // *SAP* GuiStatusPane   name="pane[4]"   Id="wnd[0]/sbar/pane[4]"
     // *SAP* GuiStatusPane   name="pane[5]"   Id="wnd[0]/sbar/pane[5]"
       // Check the event name
          char * strEvent = SAPGuiGetControlText("wnd[0]/usr/txtG_EVENT", "GuiTextField");
          RR__printf(" The Event Name: %s", strEvent);
          free(strEvent);
```

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

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
```

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.

Accessing Server Replies

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 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.

The sample script illustrates how to use DO_WSK_Recv to store the reply from the server. 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.

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 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}");
```

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

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.

Moving the Transaction Loop Statements

Moving the Transaction Loop Statements

Load testing a web site can require you to log on and log out 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 log on and log out with every transaction.

To modify the C++ 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(...), and END_TRANSACTION statements together and move them so that they are immediately above the Logout requests.

Tip: The Clear(...) statements give you the option of retaining or removing items at the end of each transaction. The Clear statements are automatically included in the script and remove all cookies, WWW cache, connections, referring page, basic authorization, and proxy authorization items. To retain any item at the end of the transaction, you must comment out the related Clear(...) command.

Visual Navigator

If you are using Visual Navigator, you can move the Transaction Loop and Transaction Cleanup statements in the tree view rather than in the C++ script. You also can select the items to clear at the end of each transaction in the Visual Navigator form view. Items you can clear or retain are: cookies, WWW cache, connections, referring page, basic authorization, and proxy authorization.

Note: By default, all objects are selected and cleared at the end of the transaction.

To modify the Visual Navigator tree view:

1. Select the Transaction Loop item and click the Move Down button in the form view until it moves below the Logon pages.
2. In the Cleared items at end of transaction section of the form view, clear the checkbox next to each item you want retained at the end of the transaction.
3. Do the same for the End Transaction by selecting the Transaction Cleanup tree item and moving it.
4. In the Cleared items area at end of transaction section of the form view, clear the checkbox next to each item you want retained at the end of the transaction. 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 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. The original script shows all objects selected in Visual Navigator for clearing at the end of the transaction. In the modified script, the Cookies are unchecked in Visual Navigator and are not cleared at the end of the transaction.

Note: When you modify the C++ script, the Clear (ALL_COOKIES) statement is manually commented out to retain cookies at the end of the transaction.

Preventing Unwanted Subrequests

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> QALoad >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 QALoad 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 QALoad 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 QALoad 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.

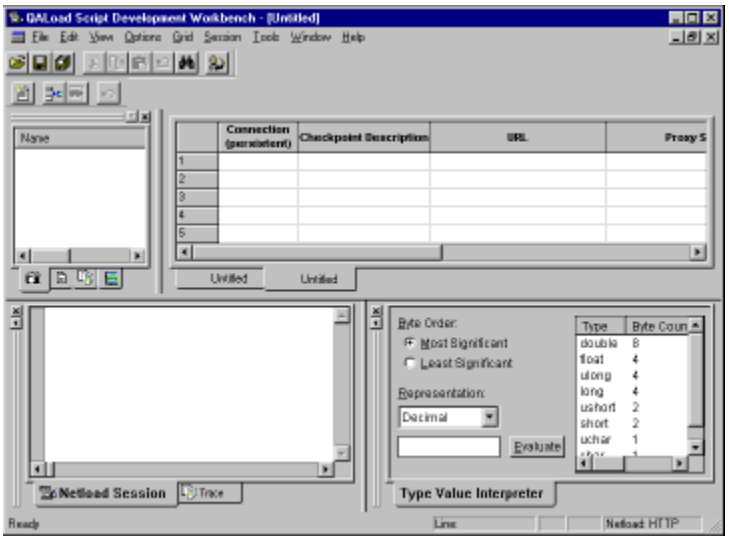
   Some fields on the grid contain pull-down menus. To activate them, click anywhere within the field. Then make your selection from the menu that appears.
5. When you are finished, select File>Save to name and save the datapool file.

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 QALoad 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. QALoad creates a C++ 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`).
3. (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

Transfer 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 QALoad 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 QALoad Script Development Workbench, click Session>QARun to start a QARun 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 QARun.
4. In the Environment field, select the appropriate QARun environment.
5. In the QARun Script field, enter the name of the QARun transaction to insert, or select it from the list, which contains a record of the last five QARun script names you entered.

   Although you can enter a script name from any database, when the test is actually running and QALoad invokes QARun, QARun attempts to retrieve that script from its default database. Therefore, in the QARun 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 QALoad to automatically insert a checkpoint into the script after this QARun transaction.
7. In the QALoad Script Name field, enter a name for this QALoad 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 QARun 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 QALoad 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 QALoad Script Development Workbench, select Session>QARun to start a QARun scripting session.
2. Select Session>New Template to create a new script from the QALoad template script.
3. In the Choose Script Name dialog box, enter a name for the new QALoad 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 QARun 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 QARun.
     - " In Environment, select the appropriate QARun environment.
     - " In QARun Script Name, enter the name of the QARun transaction to insert, or select it from the list, which contains a record of the last five QARun script names you entered. Note that you can enter a script name from any database; however, when the test is actually running and QALoad invokes QARun, QARun will attempt to retrieve that script from its default database. Therefore, in the QARun 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 QALoad script.
5. When you are finished, save any changes.
6. To compile the script for testing, select Session>Compile.

Packaging Scripts for ClientVantage

Overview - Packaging Scripts for ClientVantage

ClientVantage users must verify that versions of scripts and Player software located on each agent machine are compatible. When disparities exist, for example, when production systems are upgraded, the test scripts used to monitor the end-user experience also must be updated.

QALoad's Package Script function enables you to select a QALoad script, verify that it is up-to-date, and then package it in a zip file that you can transfer to Player machines in any location. The Package Script utility automatically determines and includes all associated files, such as datapools and binary files, that are required to run the script. The zip file contains the script, its auxiliary files, and a manifest file that lists the packaged files and the middleware the script uses.

Note: You must compile scripts before placing them in the zip file.

How to Package a Script

Use this procedure to select a QALoad script and package it and all its associated files into a zip file that you can transfer to a Player machine for use by ClientVantage.

To select a script:

1. Open a session in the Script Development Workbench.
2. Select a script in the Workspace pane and click File>Package Script. The Package Scripts dialog box appears with the script you selected displayed in the Script File field.

Note: You can open the Package Scripts dialog box before choosing a script, then click Browse to open the Select Script File to Package dialog box and select the script.

To package the script:

1. In the Package Scripts dialog box, click Package. The Select Package Directory and Name dialog box appears.
2. Click the down arrow in the Save in field to select the directory where you want to save the zip file. The default directory provided is `Program Files\Compuware\QALoad`.
3. In the File name field, type a name for the zip file. The default name provided is the name of the script.
4. Click Save. The zip file is created and saved in the directory you selected. The Package Scripts dialog box displays.

Note: You can only package compiled scripts. If you try to use the Package Scripts utility on an uncompiled script, an error message appears. You must exit the Package Scripts dialog box, compile the script, and start the script packaging process again.

5. Select another script to package, or click Close.

To unpackage the script:

To unpackage the script, access the Vantage Console in ClientVantage where you can publish the script. For more information on how to use the Vantage Console, refer to the online help in ClientVantage.

Troubleshooting

ODBC Memory Error Crash

Whenever a user-started capture is initiated, QALoad appears to begin capturing, but as soon as the program to capture against it is started, a memory exception is encountered.

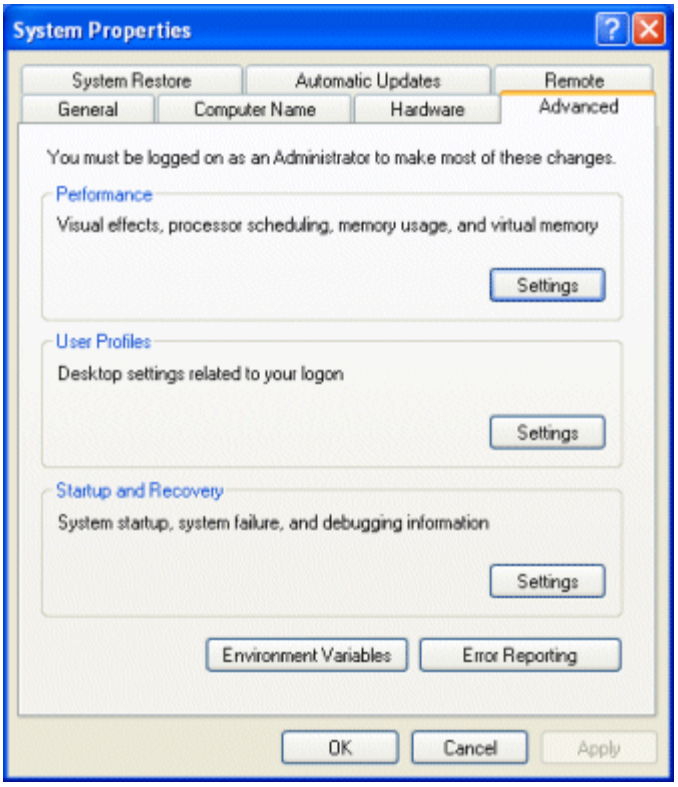
The following error is not encountered immediately. It may not occur until you attempt to click on something or perform some sort of activity on your machine.

```
Application popup: explorer.exe - Application Error : The instruction at
"0x1001ee30" referenced memory at "0x1001ee30". The memory could not be
"written".
```

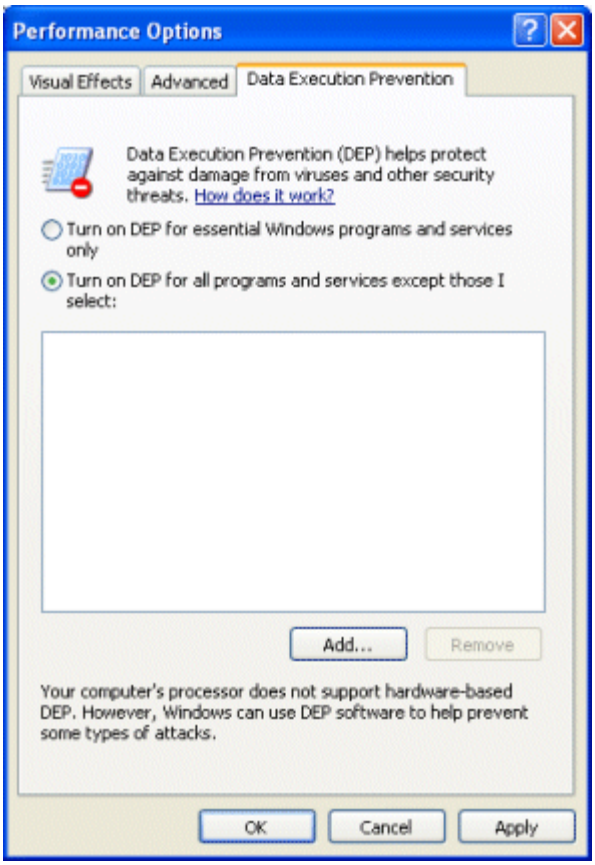
To fix the ODBC memory error crash:

If you are running Windows XP Pro SP2, perform the following steps:

1. Right-click on My Computer and select Properties. The System Properties dialog box appears. (If your My Computer is not available, access Control Panel>System.)
2. Select the Advanced tab, then click Settings in the Performance group box.



3. On the Performance Options dialog box, select the Data Execution Prevention tab. Make sure that Turn on DEP for all programs..." is selected.



4. Click Add, then navigate to select the following applications:
   a. `<systemroot>\explorer.exe`
   b. `<systemroot>\system32\rundll32.exe`
5. Click OK for any warnings that appear.
6. Click OK on the Performance Options dialog box to save your changes and close the dialog box.

The Default Session Prompt Did Not 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 QALoad 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"

Citrix

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 `CtxWaitForWindowMove` command in your script should target the final window move in the series. This issue also applies to the `CtxWaitForWindowResize` 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 `CtxWaitForWindowCreate` command may time out. If this occurs, remove the wait point or insert a `CtxWaitForScreenUpdate` command to preserve the wait in the script.

SAP

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.

SAP Script Validation Fails

If your SAP script fails during validation it perform any of the following procedures to resolve the problem.

Disable Automatic Proxy Configuration in Internet Explorer

To disable automatic proxy configuration:

You may need to disable automatic proxy configuration in Internet Explorer.

1. In Internet Explorer, click Tools>Internet Options.
2. On the Connections tab, click LAN Settings.
3. Ensure that the Use automatic configuration script check box is cleared.

Increase the Script Execution Timeout Value

To increase the timeout value:

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.

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.

Do Not Minimize the SAP Window

To maximize the SAP window:

During validation of SAP scripts, do not minimize the SAP window. If the window is minimized, the validation may fail. This problem does not occur if you do the following:

1. In Conductor, select the Script Assignment tab.
2. In the Type column, click [...] to browse.
3. Select the Hide Graphical User Interface for SAP Users option.

Compiler Errors with SAP Scripts

If you receive a type mismatch error when compiling an SAP script, you must remove the quotation marks around the last parameter of the affected command. For example:

`error C2664: '<FuncName>' : cannot convert parameter n from 'char [n]' to 'long'`

This compiler error, which can occur in commands that manipulate column widths, indicates a data type error and can be corrected by removing the quotation marks around the last parameter.

The following example shows the lines in a script that could cause an error, and the corrected version of the same lines:

Script that produces the error:

```
SAPGuiPropIdStr("wnd[0]/shellcont[1]/shell");
SAPGuiCmd2(GuiCtrlTree, SetColumnWidth, "REPNAME", "218");
```

Corrected script:

```
SAPGuiPropIdStr("wnd[0]/shellcont[1]/shell");
SAPGuiCmd2(GuiCtrlTree, SetColumnWidth, "REPNAME", 218);
```

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.

Accessing Analyze

The following procedures describe how to start QALoad Analyze.

To access Analyze from the QALoad Conductor:

1. In the QALoad 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. If you did not select the Launch Analyze After Test check box before the test, follow the steps below.

1. Click Tools>Analyze.
2. In QALoad 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.

To access Analyze from the Windows Start menu:

Click Start>Program Files>Compuware> QALoad >Analyze.

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, QALoad processing of server replies, and other QALoad 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.
- QALoad 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 and Toolbar Buttons

Click a menu or toolbar name in the following list for a description.

File

Edit

View

Template

Tools

Window

Help

Analyze Toolbar buttons

Graph Toolbar buttons

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.

You can access QALoad Analyze and open a timing file containing test results from each of the QALoad components.

To access Analyze from the QALoad Conductor:

1. In the QALoad 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. If you did not select the Launch Analyze After Test check box before the test:

1. Click Tools>Analyze.
2. In QALoad 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.

To access a previously-created timing file in Analyze from the Windows Start menu:

1. Click Start>Program Files>Compuware> QALoad >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.

To access a previously created timing file from the QALoad Script Development Workbench:

1. In the QALoad Script Development Workbench, click Tools>Analyze.
2. In QALoad 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

The QALoad Analyze Workspace displays timing file data in groups. Each group displays different aspects of the data from a timing file. The data displayed and the groups available may vary, depending on the type of data collected during the load test.

Click a group name below to view the type of data displayed in each group.

Reports	Top Processes
Checkpoints	RIP Files
Counters	ApplicationVantage
Server Monitoring	Expert User
Player Performance Counters	


perfect.tim
Reports
Summary
Session
Concurrent Users
Response Time Analysis
Output
Client Throughput
Server Monitoring
Transaction Throughput
Top Ten Longest ...
Worst Performing...
Player Performance
Expert User
ApplicationV...
Checkpoints
Counters
Server Monitoring
Player Performance Counters
Top Processes
RIP Files
ApplicationVantage
Expert User
perfect.tim

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:
     - Do NOT use a template when opening timing file - No template is applied and only the Summary report for the timing file displays.
     - 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.
     - 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 files. This closes all 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 does not exist in the current file, a dialog box appears with the name of the report or graph that cannot be displayed.

Applying Templates Globally with the Options Dialog Box

Use the Analyze Options dialog box to select template options that globally set how timing files open. Using a template saves the reports, graphs, and thresholds defined in the template.

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

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

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

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.

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

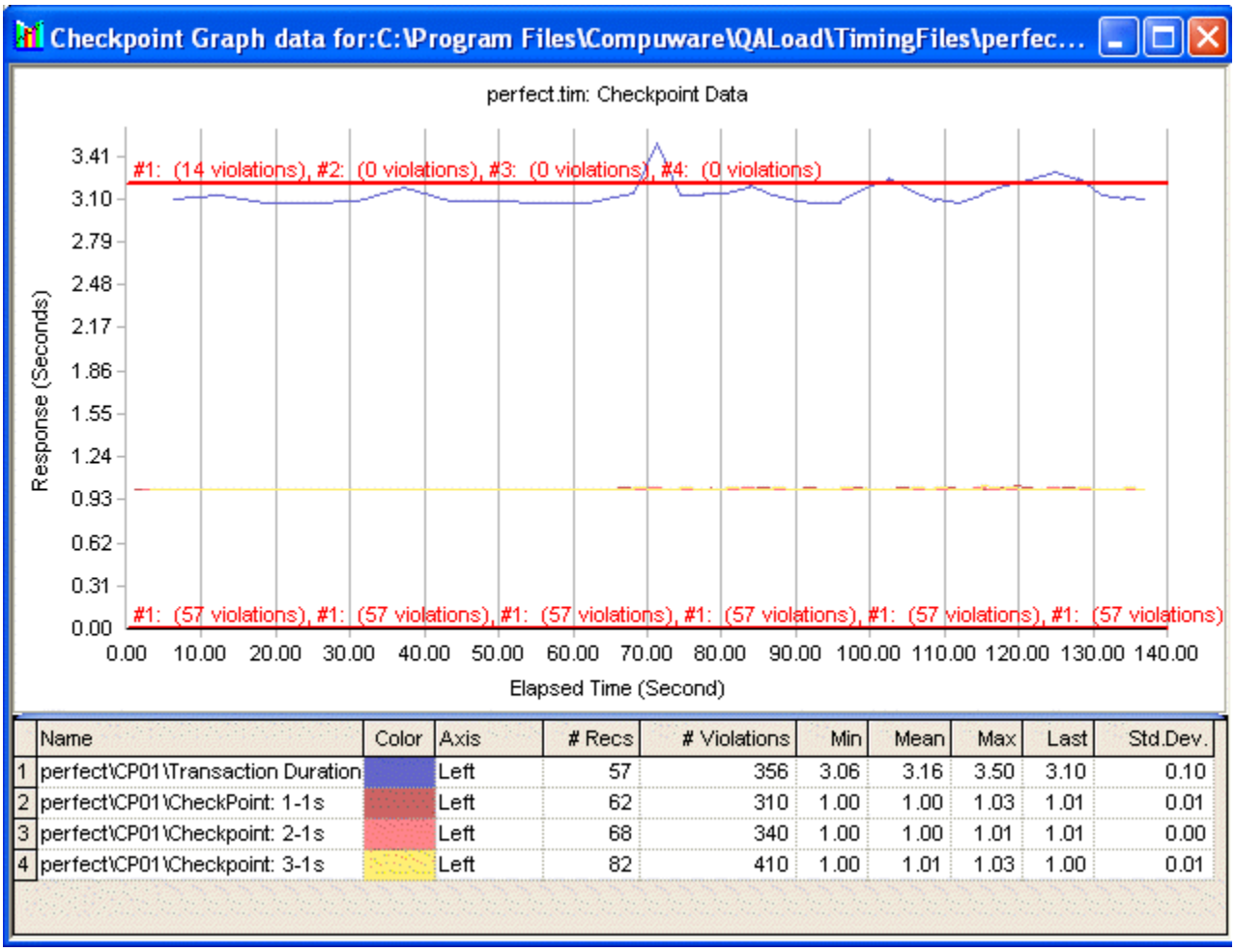
Using Thresholds

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

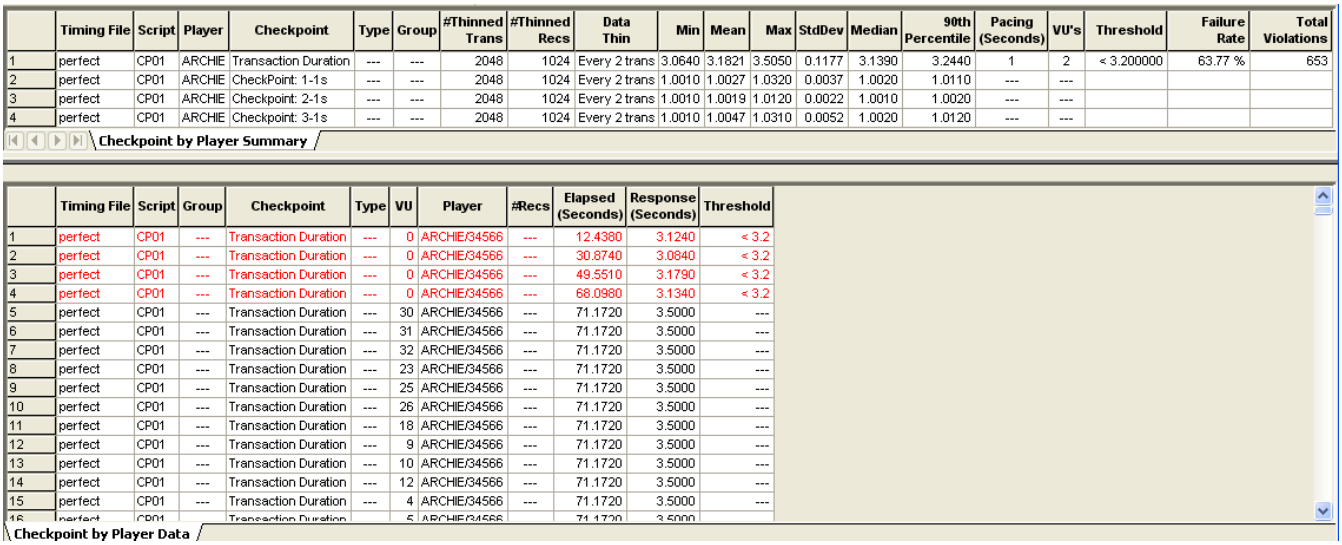
The selected data is displayed in a line graph format in the Data window. Graphs show thresholds as horizontal lines with the number of failed points.



	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	---

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.

Creating a Chart or Graph

Analyze Graph Types

The following basic graph types are available in QALoad Analyze.

Line Graph

A line graph plots response times versus elapsed times for the selected checkpoints. It provides a good representation of how much fluctuation there is in response times over the course of a test.

Bar Graph

A bar graph shows the median, mean, or percentile response times for the selected checkpoints.

Transaction Throughput Graph

This type of graph shows the cumulative number of transactions that occurred within the user-specified time range over the duration of the test.

Response Time Distribution Graph

This type of graph shows the percentage of checkpoint timings that fall within a particular response time range. A response time distribution graph shows if response times tend to fall within the range or are widely dispersed. A response time distribution graph only shows results for a single checkpoint, although it can compare results from multiple timing files.

Cumulative Response Time Distribution Graph

This type of graph shows the percentage of transactions for a single checkpoint that have a response time equal to or less than a specified value.

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.

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 thin timing data in Conductor:

1. With your test session ID file open, click the Script Assignment tab.
2. For each script for which you would like to thin your test data, click the button in the Timing Options column.
3. On the Timing Options dialog box, click the Enable Timing Data Thinning check box.
4. In the Thin Every... field, type the number of transactions to average. The average is sent to the Conductor for inclusion in the timing file, rather than every value.
5. Click OK.
6. Save your changes to your test session ID file by choosing File>Save from the Conductor menu.

For more details about the Timing Options dialog box, see Timing Options.

To set up data thinning in Analyze:

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 QALoad 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 BAR graph displays only one data point for each selected counter. The data point represents the mean, median, or percentile response time. When you select a LINE graph after a BAR graph through the BAR graph's context menu, the LINE graph displays the data of the BAR graph, not the data from the tree control on the left pane. Since there is only one data point per selected counter, the line graph does not display a line. This display is different from the LINE graph that you request from the tree control, which displays all the data points for each selected counter.

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 QALoad 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 QALoad 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 QALoad 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 QALoad 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 QALoad 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.

Creating a Scatter Chart

To create a scatter chart:

1. Click the View Graph button or choose Graph from the View menu. The Select Graph dialog box appears.
2. Select the Line graph type.
3. Click OK. Data for the selected checkpoints is presented in a line graph format in QALoad Analyze's data window.
4. Right-click anywhere in the graph, and select Gallery from the menu. The gallery of graph types displays.
5. Click the Scatter  type graph.
6. Right-click anywhere in the graph.

7. Select Properties from the menu. The Chart FX Properties dialog box displays.
8. Click the Series tab.
9. In the Show every: field, click the up or down arrow to increase or decrease the number of data points displayed, for example, show every 5th point.
10. In the Size: field, click the up or down arrow to increase or decrease the size of the scatter points.
11. Click Apply.
12. In the Shape: field, click the arrow and select the shape for the scatter points.
13. Click Apply, then click OK.

See Customizing a graph for more information about customizing a graph.

Creating Financial Charts

There are three financial charts in QALoad Analyze: Candlestick, High-low-close, and Open-high-low-close. Use this procedure to display these charts in the graph Gallery, and then create graphs of these types.

To access the financial charts:

1. Go to the QALoad root directory. The default is `c:\Program Files\Compuware\QALoad`.
2. In a text editor, open the graphprops.xml file.
3. In the line `<FinancialCharts value="false/">`, replace the value by highlighting the word **`false`** and typing **`true`**.
4. Click File>Save, then click File>Exit.

To use the financial charts:

In QALoad Analyze, do the following:

1. Click the View Graph button or choose Graph from the View menu. The Select Graph dialog box appears.
2. Click the down arrow select the Line graph type.
3. Click OK. Data for the selected checkpoints is presented in a line graph format in QALoad Analyze's data window.
4. Right-click anywhere in the graph.
5. Select Gallery from the menu. The gallery of graph types displays.
6. Click the graph you want to use.

Customizing a Chart or Graph

Customizing a Graph

Change the style and appearance of a graph using options available from either of the Graph toolbars. The Graph toolbars contain buttons for standard Windows operations as well as for customizing a graph's appearance. Display the Graph toolbars by right-clicking in an open area of a graph and choosing Toolbars>Toolbar or Annotate Toolbar from the shortcut menu.

The following features can be customized from the Graph toolbar. Click on any feature in the following lists for additional information or instructions:

Graph Type

Color

Grid Orientation (horizontal and vertical)

Legend Box

Dimension (3D or 2D)

Rotation

Z-Cluster

Font

Text/Object

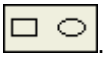
Adding Text or an Object to a Graph

To create explanatory text or an object on a graph, use any of the text or object buttons on the Graph Annotate toolbar.

To add a note or object to a graph template:

1. Display the Annotate toolbar.
2. Add text and/or an object to the graph.

Storing a Note or Object in a Template

You can add explanatory text or objects to a graph to be utilized as part of a template when the template is applied to future timing files. For text, use the Text Box or the Balloon with Text button  . To create an object, use the Rectangle or Circle button  .

Note: These are the only objects that can be stored in a template.

To save the information in a template:

1. Complete the steps above for adding a note or object.
2. Follow the instructions for Creating a New Template. The saved note or object is available when the template is applied to future timing files.

Viewing 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

Application Vantage

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.

With the Summary Report open in Analyze, click a heading in the Test Information area to display detailed information on scripts, errors, and messages. When you enable data thinning, the number of errors and messages is thinned, accordingly.

Note: There is a limit of 3000 errors processed for each script and 3000 messages processed for each group within a script. When you click # Errors or #Messages for detailed information, any script or group within the script that exceeds the limit does not display. A script or group can fall within the limit when data is thinned but exceed the limit when data is not thinned. This means that more scripts may appear in the Error section and more groups may appear within scripts in the Script Message section when data is thinned.

Sample Summary Report

For a brief description of each report section, scroll down and click a section heading 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

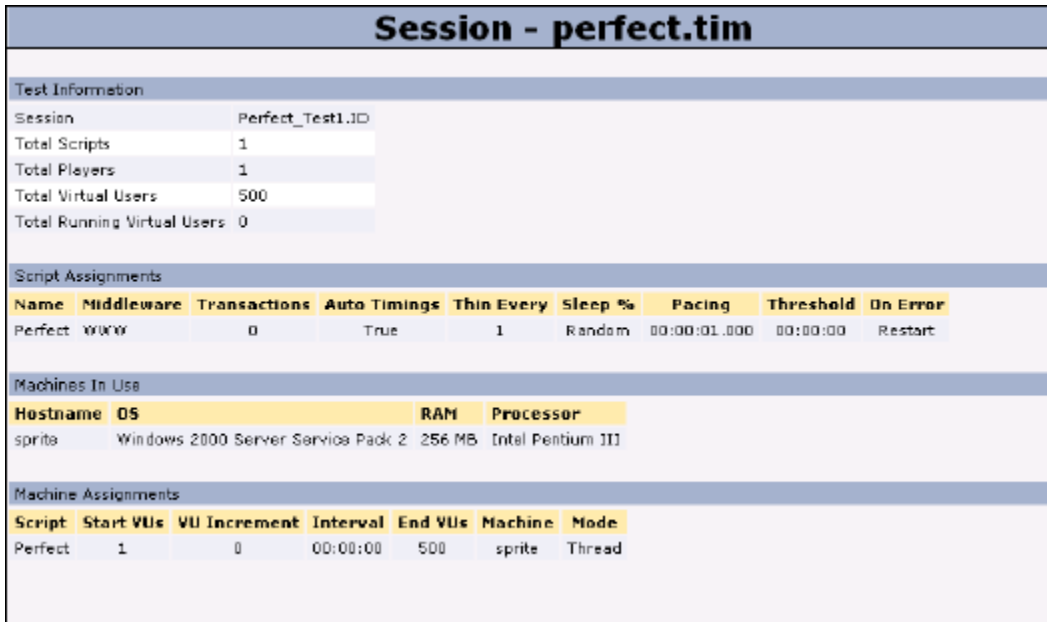
Hostname	OS	RAM	Processor
dtw114910d01	Windows XP Workstation Service Pack 1	1023 MB	Intel Pentium 4

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

For descriptions of the information provided in each section, click the sections in the following image.



Session - perfect.tim

Test Information

Session	Perfect_Test1.JD
Total Scripts	1
Total Players	1
Total Virtual Users	500
Total Running Virtual Users	0

Script Assignments

Name	Middleware	Transactions	Auto Timings	Think 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

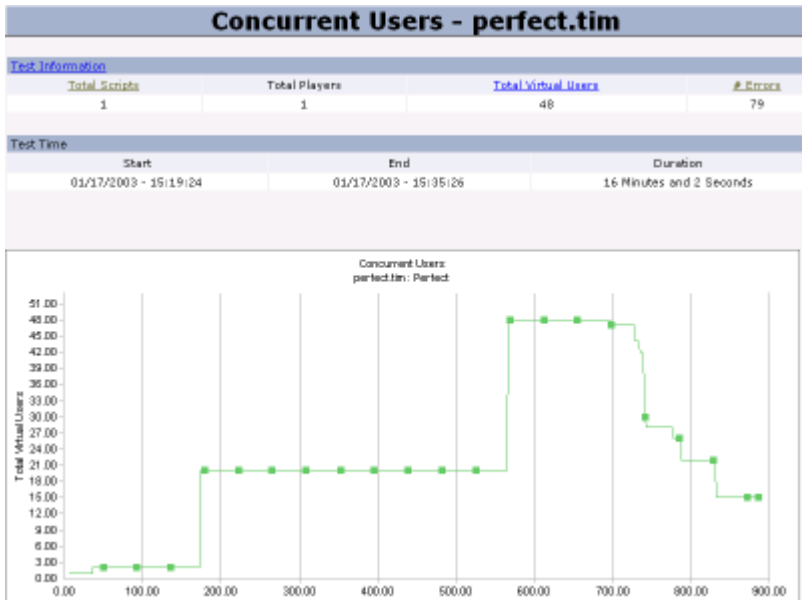
Machine Assignments

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

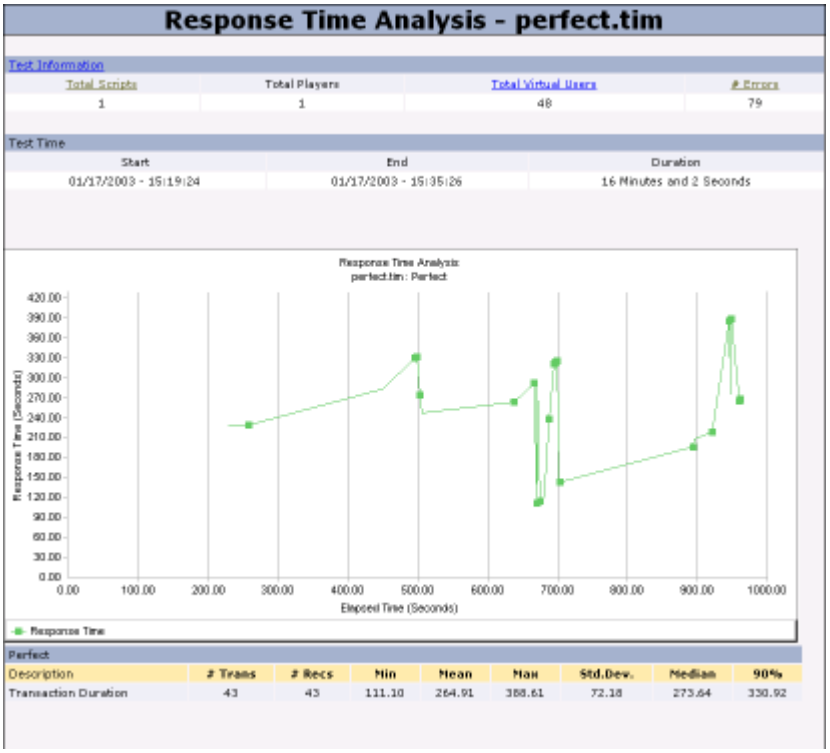
Test Time

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:

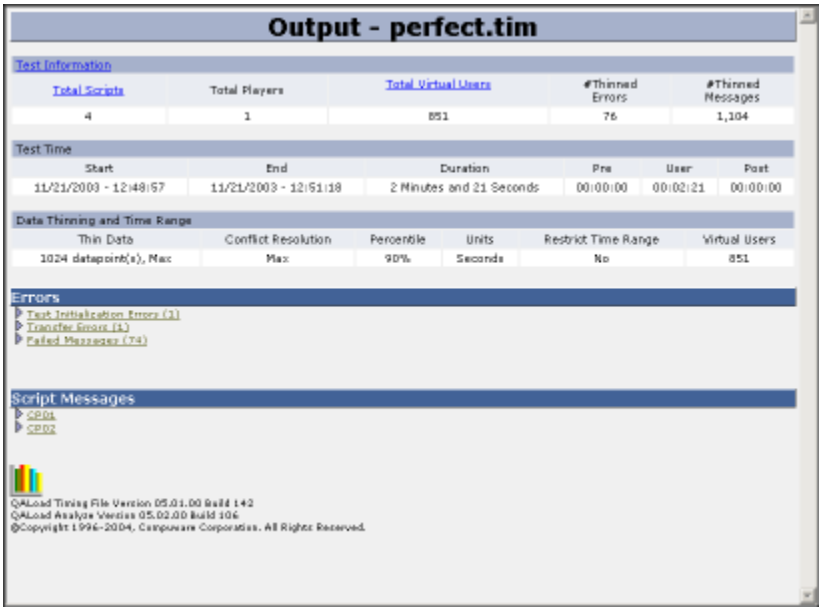
#Trans: Number of data points used to calculate the statistics.
#Recs: Number of data records. This value, if different from the value of #Trans, reflects the number of checkpoint records that are used for analysis after data thinning has been applied.
Min: Minimum recorded response time.
Max: Maximum 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.



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.

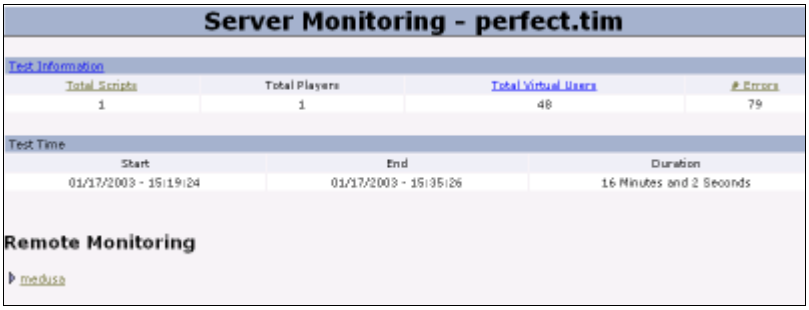


Client Throughput report

Provides a graph of HTTP Reply analysis for key HTTP counters, HTTP counter vs. elapsed time.

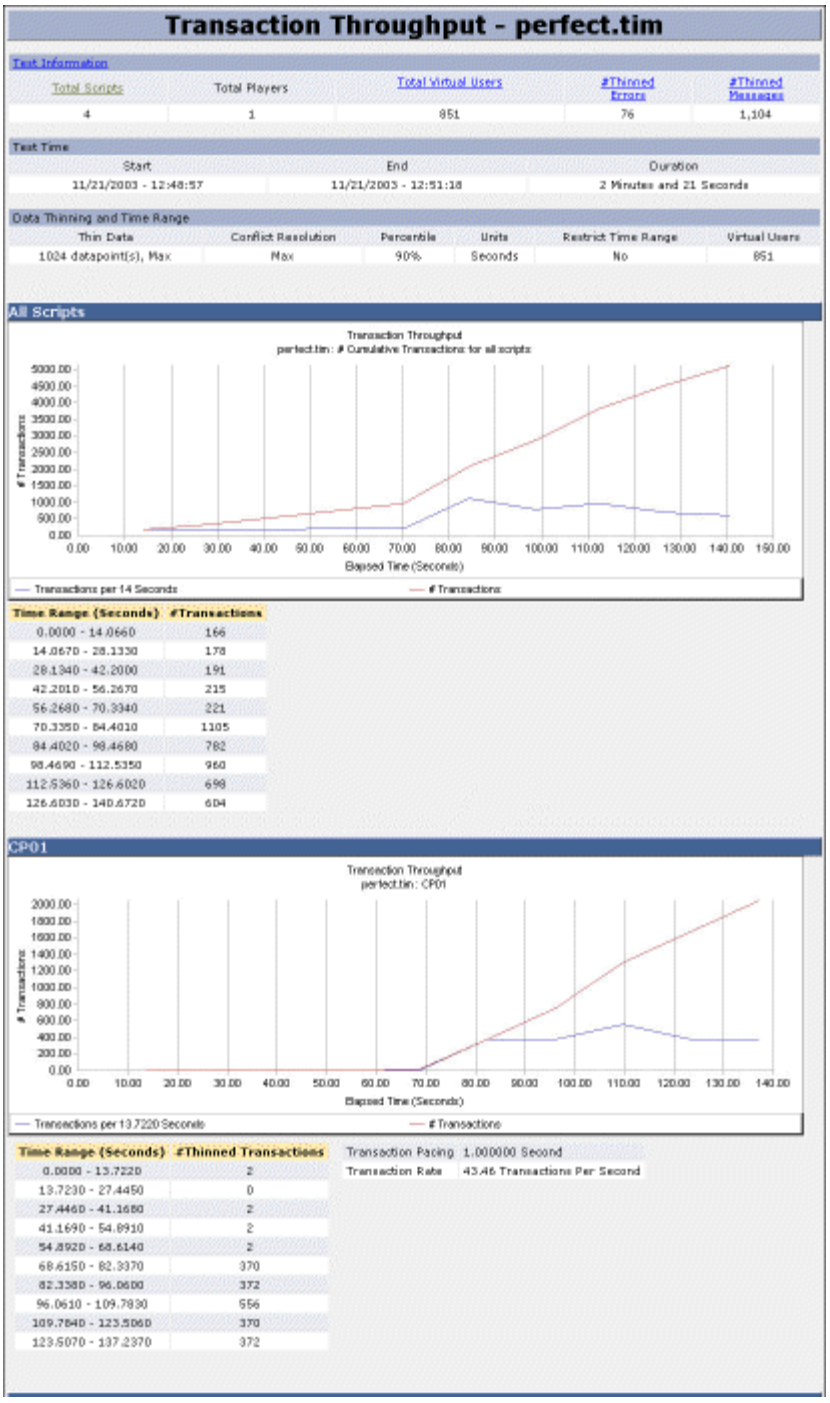
Server Monitoring report

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.



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

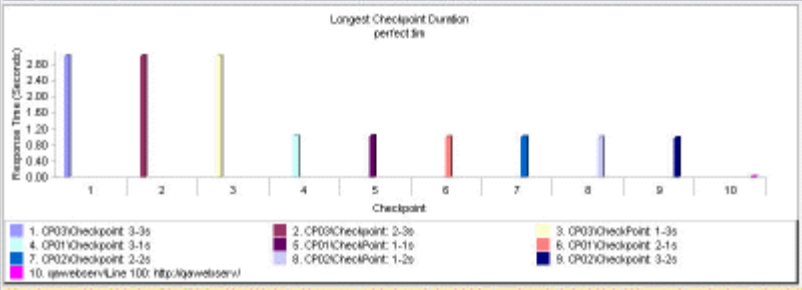
Test Time

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

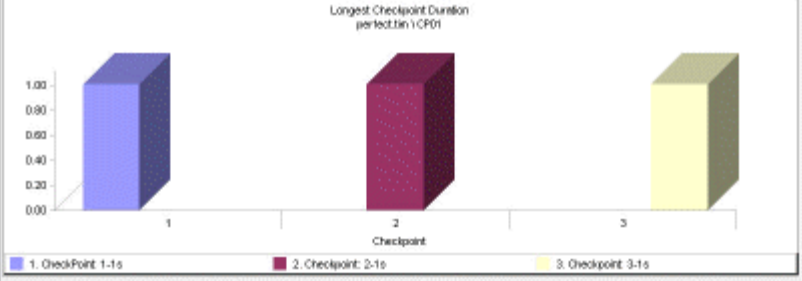
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

Longest Checkpoint Duration Summary



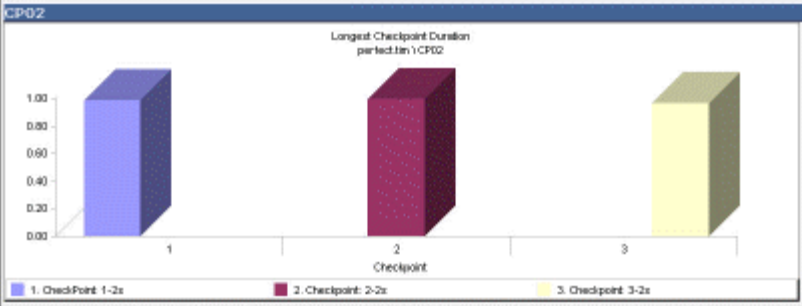
	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1	CP03\Checkpoint: 3-3s	1,024	75	3.0040	3.0054	3.0240	0.0035	3.0040	3.0050
2	CP03\Checkpoint: 2-3s	1,024	74	3.0040	3.0048	3.0240	0.0026	3.0040	3.0050
3	CP03\CheckPoint: 1-3s	1,024	82	3.0040	3.0046	3.0150	0.0020	3.0040	3.0050
4	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

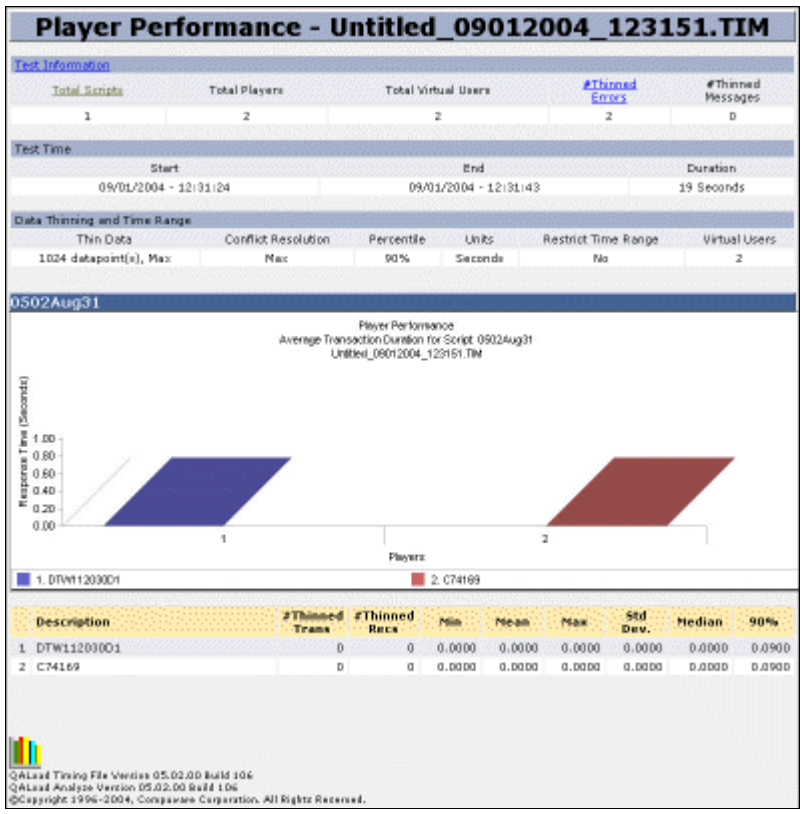


	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1	CheckPoint: 1-2s	1,024	839	0.0000	0.9883	2.0030	0.5760	0.9810	1.7830
2	Checkpoint: 2-2s	1,024	819	0.0100	0.9924	2.0030	0.5679	1.0020	1.7650
3	Checkpoint: 3-2s	1,024	832	0.0100	0.9662	2.0030	0.5956	0.9220	1.8130

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

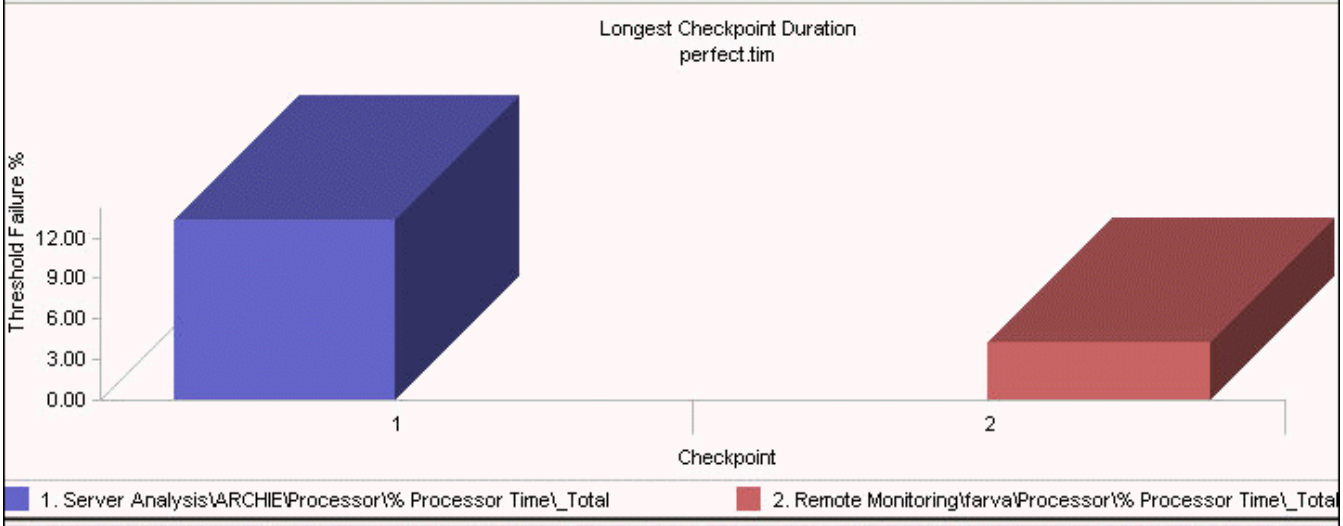
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

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)

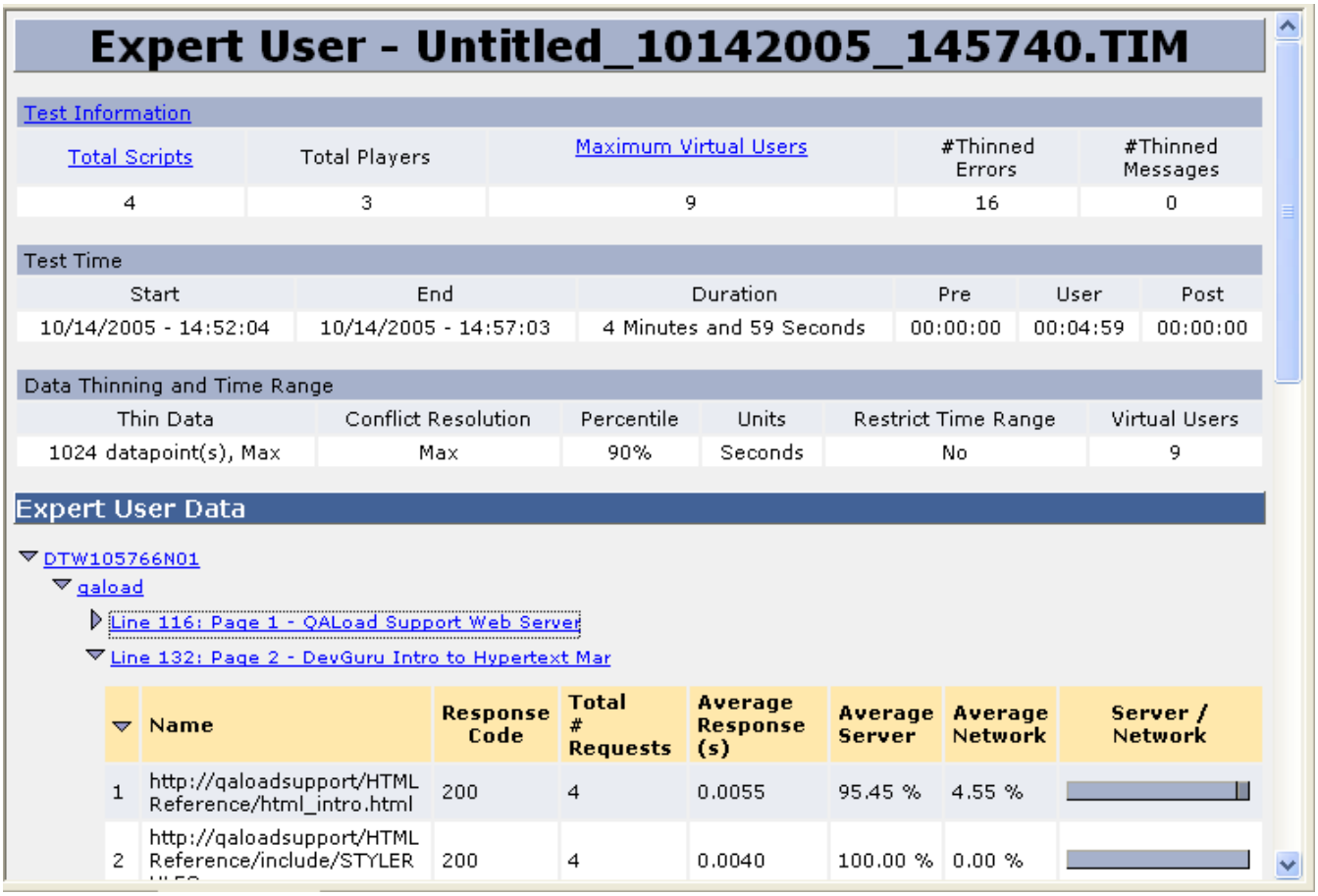
Expert User Report

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.



Expert User - 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	Pre	User	Post
10/14/2005 - 14:52:04	10/14/2005 - 14:57:03	4 Minutes and 59 Seconds	00:00:00	00:04:59	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	9

Expert User Data

DTW105766N01
qaload
Line 116: Page 1 - QALoad Support Web Server
Line 132: Page 2 - DevGuru Intro to Hypertext Mar

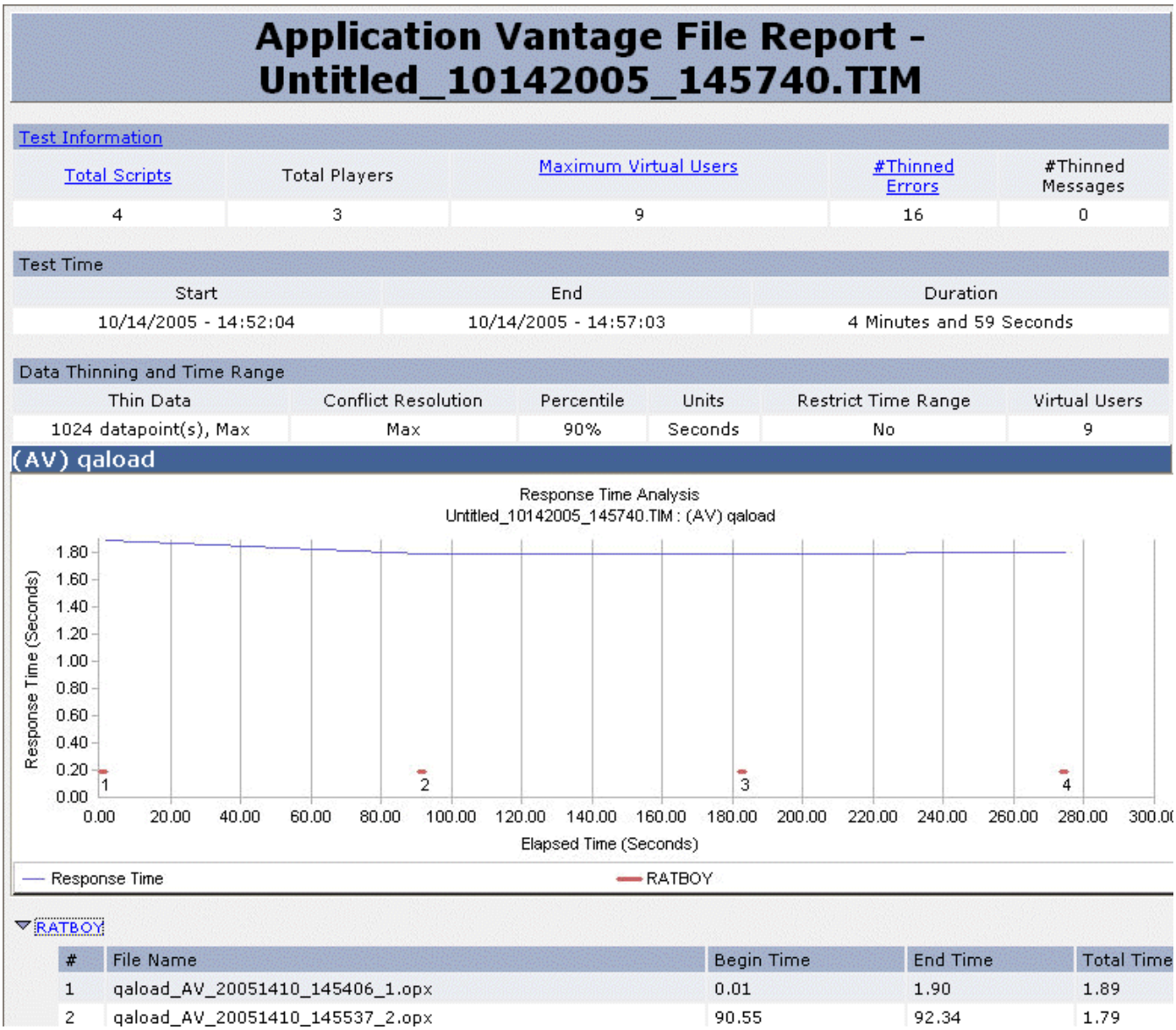
	Name	Response Code	Total # Requests	Average Response (s)	Average Server	Average Network	Server / Network
1	http://qaloadsupport/HTML Reference/html_intro.html	200	4	0.0055	95.45 %	4.55 %	
2	http://qaloadsupport/HTML Reference/include/STYLER	200	4	0.0040	100.00 %	0.00 %	

ApplicationVantage Report

Displays information on the scripts that ran on each player machine. It contains a summary section with overview information about the test, and a graph of the transaction response time of the script in seconds.

Note: If Expert User data was collected (WWW only), the Network and Server percentage data also is included.

A detail section displays the beginning and ending time for the trace files produced by the scripts running on the ApplicationVantage player machines. A separate section appears for each ApplicationVantage Player machine that ran during the test period, .



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.

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 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 on the Workspace tab of 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 that 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 requires 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 that 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

D