*QA*Load

Script Development Guide

Release 05.01

Please direct questions about *QA*Load
or comments on this document to:

***QA*Load Customer Support**
Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

1-800-538-7822

Outside the USA and Canada, please contact
your local Compuware office or agent.

Doc. CWQLRX510
April 06, 2004

Table of Contents

Part 3:
Customizing a Test Script

Introduction

This guide is divided into the following parts:

- **Part 1: Getting Started** — This section provides overview and introductory material about the *QA*Load Script Development Workbench and the script development process.
- **Part 2: Developing a Test Script** — This section includes important information you may need to know before recording a script, describes the procedure for recording a test script, and explains the basic components of a *QA*Load test script.
- **Part 3: Customizing a Test Script** — This section details the various methods you can use to customize a test script to account for special situations at playback, such as variable data.

Who Should Read This Guide?

The *QALoad Script Development Guide* is intended to provide procedural information for creating test scripts for your application. It is designed to guide you through the preparation of a test script, including recording a transaction, converting it to a reusable test script, and modifying the code to accommodate variable information and other special circumstances. This guide does not contain product overview information or procedures outside the realm of script preparation, such as setting up a test or analyzing test results. For overview information and general test procedures, refer to your *QALoad Testing User's Guide*.

If you have not yet reviewed the *QALoad Testing User's Guide*, we recommend you do so before using this guide to create a test script. The *QALoad Testing User's Guide* can familiarize you with the product and assist you in preparing for a load test.

When you are comfortable with the *QA*Load testing process, use this *QALoad Script Development Guide* to prepare your test script(s).

Product Enhancements

For a detailed listing of product enhancements made in this release, refer to the *Release Notes*.

Related Publications

In addition to this guide, the *QA*Load documentation set includes the following related publications:

- *QALoad Testing User's Guide* introduces you to the load testing process and provides procedures for running and analyzing tests. It also provides reference information for the product's UNIX Player utilities.
- *QA*Load's online help facilities provide field-level and overview information for the *QA*Load product screens. The online help also includes the *QA*Load Language Reference, which provides syntax definitions, parameter descriptions, and examples for all commands that are available for use in *QA*Load scripts.
- The *QACenter Installation and Configuration Guide* includes system requirements and instructions for installing *QA*Center products.
- The *Distributed License Management Installation Guide* includes instructions for licensing your *QA*Center products.
- *Release Notes* details system requirements for using *QA*Load, enhancements made to the product for this release, technical information that may affect how you use the product, any known issues related to using the product, and customer support contact information.

Typographical Conventions

The *QA*Load documentation set uses the following typographical conventions:

Description	Examples
Window controls (buttons, menu items, etc.) are shown in **bold type**.	Click **OK**. Select **File>New**.
A fixed pitch font is used for script examples and error messages.	`BEGIN_TRANSACTION();`
Items in angle brackets indicate placeholders for information you supply.	`<userid>, <password>`

World Wide Web Information

To access Compuware Corporation's site on the Internet World Wide Web, point your browser at http://www.compuware.com. The Compuware site provides a variety of product and support information.

FrontLine Support Web Site

You can access online technical support for Compuware products via our FrontLine support Web site. FrontLine provides you with fast access to critical information about your *QA*Center product. You can read or download documentation, frequently-asked questions, and product fixes, or directly e-mail Compuware with questions or comments.

In order to access FrontLine, you must first register and obtain a password. To register, point your browser at http://frontline.compuware.com.

Getting Help

At Compuware, we strive to make our products and documentation the best in the industry. Feedback from our customers helps us maintain our quality standards. If you need support services, please obtain the following information before calling Compuware's 24-hour product support hotline:

- The release (version), and build number of your *QA*Load product. This information is displayed when you select the About command from the Help menu. The name and release are also on the covers of the product documentation.
- Installation information, including installed options and whether it is installed in the default directories.
- Environment information, such as the operating system and release on which the product is installed, memory, hardware/network specifications, and the names and releases of other applications that were running.
- The location of the problem in the *QA*Load product software, and the actions taken before the problem occurred.
- The exact product error message, if any.
- The exact application, licensing, or operating system error messages, if any.
- Your Compuware client, office, or site number, if available.

***QA*Load Technical Support**
Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

1-800-538-7822

Part 1:
Getting Started

Chapter 1. Overview

This chapter contains the following sections:

- **Accessing the QALoad Script Development Workbench —** Directions for accessing the *QA*Load script development facilities.
- **The QALoad Script Development Workbench Main Window —** Description of the *QA*Load Script Development Workbench main window and the purpose of each pane.

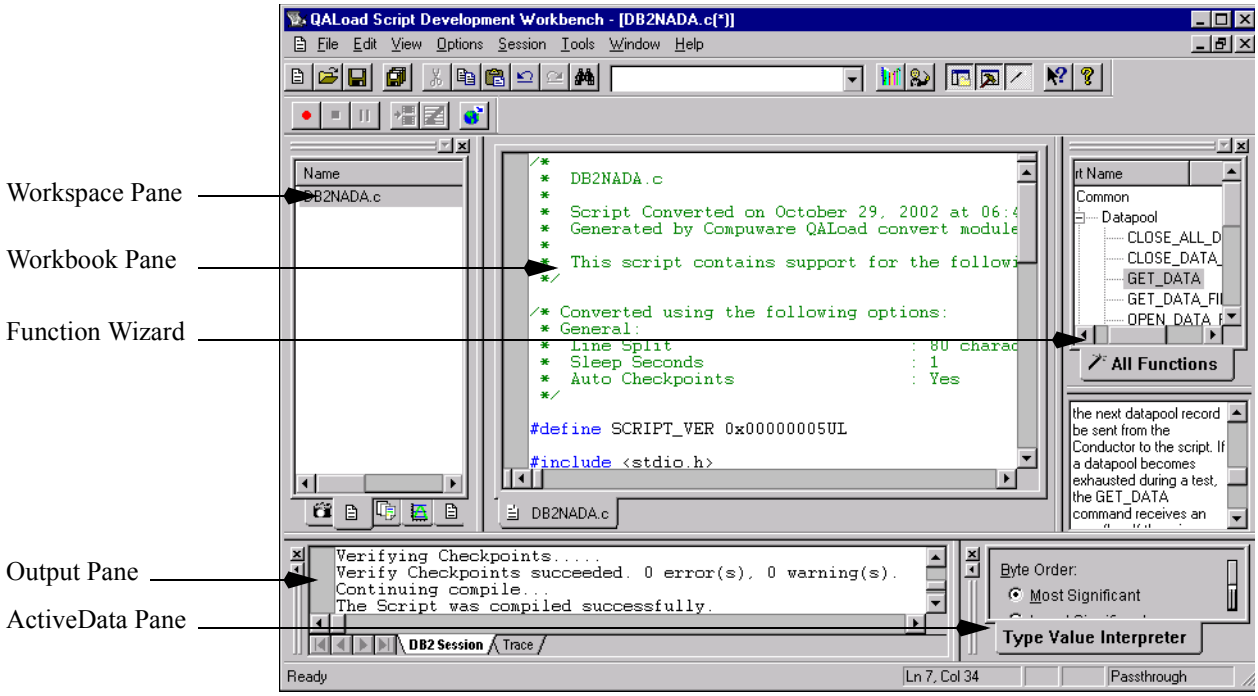
Accessing the *QA*Load Script Development Workbench

1. Click the **Start** button and point to **Programs>Compuware>QALoad** from the Start menu. Then, select the **Script Development Workbench** icon.
2. Start your appropriate middleware or protocol session by selecting it from the **Session** menu.
3. If this is your first time accessing the *QA*Load Script Development Workbench, the Default Session Prompt opens. Set the following options:
   a. To make the open middleware session the default session every time you open the *QA*Load Script Development Workbench, select the **Make this my default Session** check box.
   b. When the **Enable default Session checking** check box is selected, every time you open the *QA*Load Script Development Workbench it will do one of the following:
      — automatically open the middleware session that was previously designated as the default session
      — or, if you did not previously designate a default session, open the Default Session Prompt so you can designate a default middleware session.

      If you do not want to designate a default middleware session, clear the **Enable default Session** check box.
4. Click **OK**.

The *QA*Load Script Development Workbench Main Window

The *QA*Load 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. Each pane is described below:



Workspace Pane: Lists the scripts, capture files, datapool files, timing files, .rip files, and log files available for the open middleware or protocol session. To access a file, click on the appropriately named tab (for example, Scripts). Then double-click the file you wish to open. The file opens in the Workbook Pane. To access a popup menu of commands available from the Workspace pane, highlight a file and right-click anywhere in the pane.

Workbook Pane: Displays the currently open file. You can modify scripts, datapool files, and capture files in this pane. To access a popup menu of commands available from this pane, right-click anywhere in the pane.

Output Pane: Displays debug and error messages. To access a popup menu of commands available from this pane, right-click anywhere in the pane.

ActiveData Pane: Interprets and displays data types as you highlight data in the Workbook Pane. Note that, by default, this pane is hidden. To view the ActiveData pane, select **View>ActiveData** from the menu.

Function Wizard: Lists all functions that are valid to use in the open script. Functions are grouped in logical sections within the top window of the wizard. When you highlight a function in the top window of the wizard, the lower window will list a description of that function and it's parameters. The Function Wizard has drag-and-drop functionality to make scipt editing a breeze. The Function Wizard opens automatically when you open a

script it is compatible with. For more details about using the Function Wizard, refer to the *QA*Load Script Development Workbench online help when you are working in a script.

For a description of the toolbar buttons available from the *QA*Load Script Development Workbench, refer to the online help.

Part 2:
Developing a Test Script

Chapter 2. Before You Begin

This chapter details steps you should take to prepare *QA*Load and the application under test to record a test script. You must set options in the *QA*Load Script Development Workbench to determine the behavior of *QA*Load and, depending upon the middleware application or protocol you are testing, you may also need to perform additional steps to prepare your application or environment for testing.

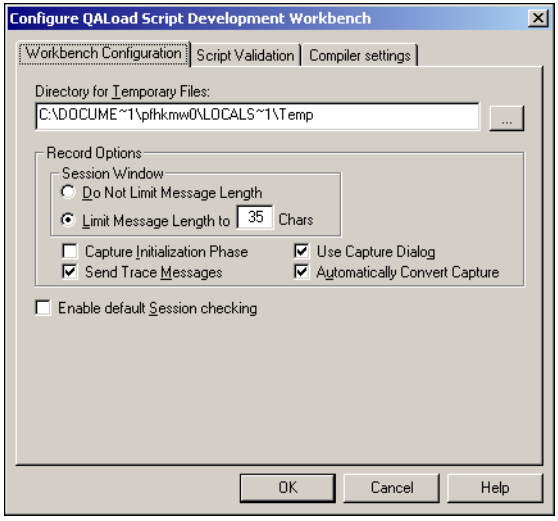
The topics included in this chapter are listed below. Please read this list carefully to determine which topics apply in your testing situation. Topics that *only* apply to a specific middleware or protocol list that middleware or protocol in the topic title. If you are not testing the specified middleware or protocol, you can ignore those topics. If a topic does not specify a middleware or protocol in the title, it applies to all testing situations and you should follow the directions in the procedure before attempting to record a test script.

- **Configuring the QALoad Script Development Workbench —** Steps you should take to determine the behavior of the *QA*Load Script Development Workbench the first time you use it.
- **Setting Recording Options —** Steps you should take to determine behavior specific to recording from your application.
- **Setting Conversion Options —** Steps you should take to set options to determine how the *QA*Load Script Development Workbench should convert your recorded transaction into a script for load testing.
- **WWW/SSL: Preparing to Record SSL Requests —** Steps you should take to configure your Web browser and prepare certificates before recording SSL requests from Web sites requiring a client certificate.
- **Tuxedo: Setting Environment Variables —** Environment variables you must set before you can successfully record a Tuxedo-based script.
- **SAP: Preparing to Record a Script —** Option you must select before you can successfully record a SAP-based script.

Configuring the *QA*Load Script Development Workbench

The first time you use the *QA*Load Script Development Workbench you should set options to determine a working directory *QA*Load can use for temporary files, compiler settings, and other general options related to the behavior of the *QA*Load Script Development Workbench. For more detailed descriptions of the fields in this procedure, press F1 from the Configure Script Development Workbench dialog box.

1. In the *QA*Load Script Development Workbench, open the appropriate middleware or Universal session.
2. From the **Options** menu, select **Workbench** to open the Configure QALoad Script Development Workbench dialog box.



3. On the **Workbench Configuration** tab, set the following options:
   a. In the **Directory for Temporary Files** field, enter or browse for a directory *QA*Load can use as a working directory for temporary files, as necessary.
   b. In the **Session Window** area, determine the length of messages appearing in the Output Pane while you are recording. If you choode to limit the length, select the **Limit Session Window...** option and then enter the maximum number of characters to display per message. If you notice a delay in your application, Compuware recommends limiting the number of characters sent to the session window.
   c. The initialization phase is the time between when the application starts and when the first window displays. Select the **Capture Initialization Phase** check

box to record database login commands in the initialization phase of applications created by PowerBuilder and similar applications.

You must have administrative access to use this feature. This option works with the User Started recording option. (It has no affect if you use the Automatic startup option.) You must use Windows NT (NT requires rebooting in order for changes to this option to take effect), Windows 2000 or XP Enterprise edition.

d. Determine whether to send trace messages back to the *QA*Load Script Development Workbench while recording. If you don't wish to send trace messages, clear the **Send Trace Messages** check box.

e. Select the **Use Capture Dialog** check box to enable a floating toolbar, while recording, that you can use to control the recording process.

f. Select the **Automatically Convert Capture** check box for *QA*Load to automatically convert your completed capture file to a C-based script immediately after you stop recording.

If you do not select this check box, you will have to manually convert your scripts using the instructions in the *QA*Load Script Development Workbench online help.

g. Select the **Automatically Compile Scripts** check box for *QA*Load to automatically compile your scripts on the selected compiler immediately after conversion.

If you do not select this check box, you will have to manually compile your scripts using the instructions in *QA*Load Script Development Workbench online help.

h. Select the **Enable default session checking** check box for *QA*Load to prompt you to set a default session the next time you start the *QA*Load Script Development Workbench.

4. On the **Script Validation** tab, set the following options for validating scripts. You can change these options at any time:

a. Select the **Automatically Recompile** check box for the *QA*Load Script Development Workbench to compile the script before running it.

b. (Java scripts only) Select the **Ask for Automatic....** check box to be prompted to validate a Java script after compiling it.

c. Select the **Only Display Player Output on Script Failure** check box to display Player messages when the script fails, but not when it runs successfully.

d. In the **Wait up to** field, type a value in seconds the *QA*Load Script Development Workbench can wait for the script to begin before timing out.


Note

Note for SAP and Citrix: Due to the time required to logon to the server, you may need to increase the timeout value to 100 seconds or more, depending on your particular setup. Set the timeout value to 100 seconds or to the length of the capture (in seconds), whichever is greater.

e. .Select the **Abort on Error** check box to stop script execution upon encountering an error.

f. Select the **Debug Data** check box to see a debug message displaying each command as it executes.

g. In the **Run As** area, choose whether to run the script as thread- or process-based.

h. In the **Number of users** field, type how many virtual users to assign to validate a script.

i. In the **Transactions** field, type the number of transactions to run.

j. In the **Sleep Factor %** field, type the percentage of each DO_SLEEP (pause in the script) to maintain for validation. The value can be a percentage between 0 and 100. The default is 0.

5. On the **Compiler Settings** tab, set options related to script compilation:

a. Select the **Automatically Recompile** check box for the *QA*Load Script Development Workbench to automatically recompile your script before running it.

b. Select the **Prompt before overwriting script** check box to be prompted if you attempt to overwrite a script with a script of the same name.

c. Select the **Verify Checkpoints** check box to automatically verify the syntax of checkpoints in your script every time your script is compiled. Note that you can also verify checkpoints manually at any time while your script is open in the editor by choosing the menu command Session>Verify Checkpoints.

d. In the **Compile Timeout** field, type the length of the compile timeout in this field. Normally, you would accept the default (120 seconds); however, you may need to increase this time if your scripts frequently fail to compile.

e. In the **C/C++** area, choose the compiler you wish to use to compile your scripts.

f. (Java scripts only) In the **JAVA_HOME** field, navigate to the root directory of your JDK. This is where the *QA*Load Script Development Workbench will look for Java resources.

g. (Java scripts only) The **JVM.DLL** field displays the directory where the JVM.DLL is located. If more than one is listed, select the appropriate one for the*QA*Load Script Development Workbench to use when validating scripts.

h. (Java scripts only) The **Workbench Classpath** field lists the part of the classpath that is common to all *QA*Load Script Development Workbench sessions that use Java. Each session, the Workbench appends required classes and JAR files to this classpath. The default is <QALoad>\Scripts;<QALoad>>\Classes;<QALoad>\Classes\qaloadbasescript.jar, explained as follows:

- \Scripts — The default direcotry where the script JAR files will be placed. When you compile a script, class files will be placed in or under this directory. This is also the default working directory where the JVM is started. Resources using relative paths should be placed in or under this directory.
- \Classes — This is where class and JAR files used by *QA*Load are located.
- qaloadbasescript.jar — If you compile outside of the *QA*Load Script Development Workbench, this file must be included in the classpath.

6. Click **OK** to save your settings or **Cancel** close without making any changes.

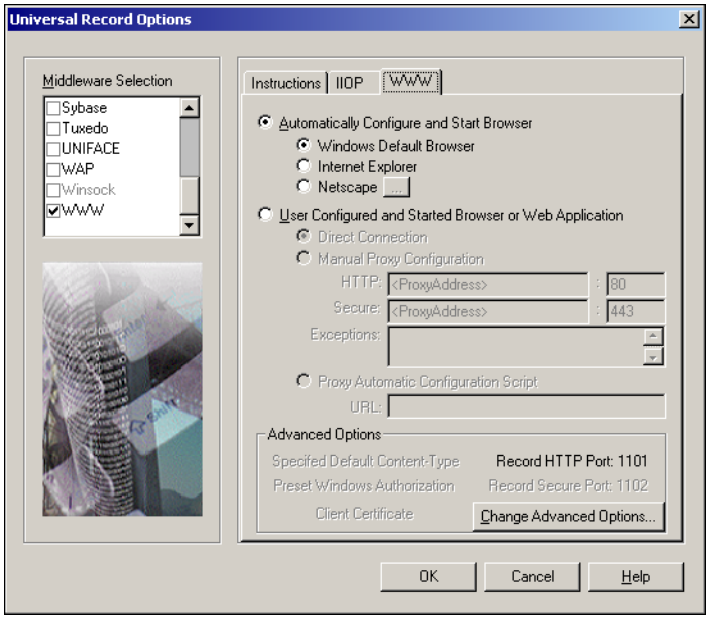
Setting Recording Options

Before you begin to record using the *QA*Load Script Development Workbench, set recording options to determine behavior specific to recording from your application.

1. From the *QA*Load Script Development Workbench **Session** menu, select your middleware or protocol to start the appropriate session, or select Universal to start a Universal session.

2. From the **Options** menu, select **Record**. The Record Options wizard opens.

3. The **Middleware Selection** area shows your selected middleware. A tab opens for each selected middleware to the right of the Middleware Selection area (see the example that follows).



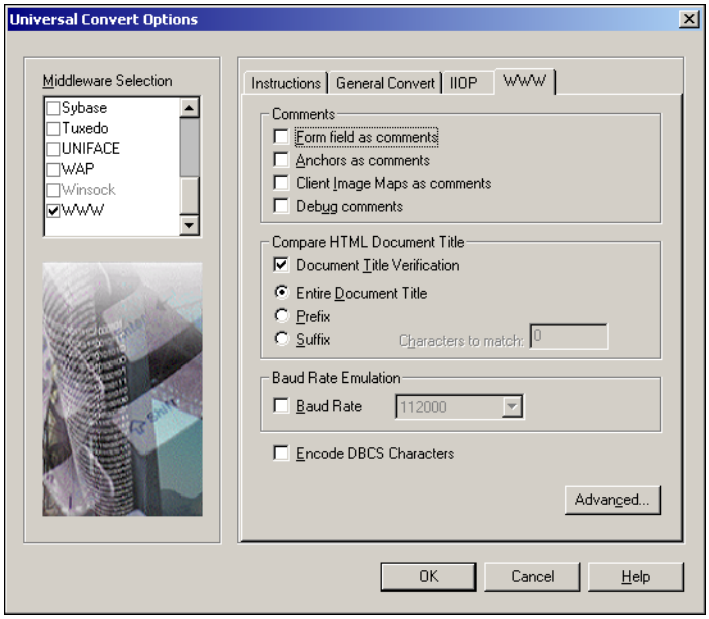
4. On the middleware tab, set the appropriate recording options. For a description of each option, press F1 or click the Help button to access online help for the wizard.
5. When you are finished, click **OK**.

These options will remain in effect until you change them.

Setting Conversion Options

Use the following procedure to set options for converting a capture file recorded from the *QA*Load Script Development Workbench into a C- , C++, or Java-based script for load testing.

1. From the *QA*Load Script Development Workbench **Session** menu, select your middleware or protocol to start the appropriate session, or select Universal to start a Universal session.
2. From the **Options** menu, select **Convert**. The Convert Options wizard opens.
3. The **Middleware Selection** area shows your selected middleware. For each selected middleware, a tab opens behind the General Convert tab to the right of the Middleware Selection area (see the example that follows)



4. On the middleware tab, set the appropriate conversion options. For a description of each option, press F1 or click the Help button to access online help for the wizard.
5. When you are finished, click **OK**.

These options will remain in effect until you change them.

WWW/SSL: Preparing to Record SSL Requests

Before you can use EasyScript for Secure WWW to record SSL requests from a Web site requiring a client certificate, you may need to perform following tasks:

- disable your browser's TLS security (Internet Explorer 5.0). See "Disabling TLS Security in Internet Explorer 5.0" on page 2-8.
- prepare a client certificate. See "Preparing SSL Certificates" on page 2-8.

Disabling TLS Security in Internet Explorer 5.0

If you will be recording SSL requests using Internet Explorer 5.0, do the following to ensure that TLS 1.0 is not enabled in your browser:

1. From the Internet Explorer **Tools** menu, select **Internet Options**.
2. On the **Advanced** tab, scroll to the **Security** section.
3. Make sure that the option **Use TLS 1.0** is not selected.
4. Click **OK**.

Preparing SSL Certificates

You can prepare a client certificate for SSL testing two ways:

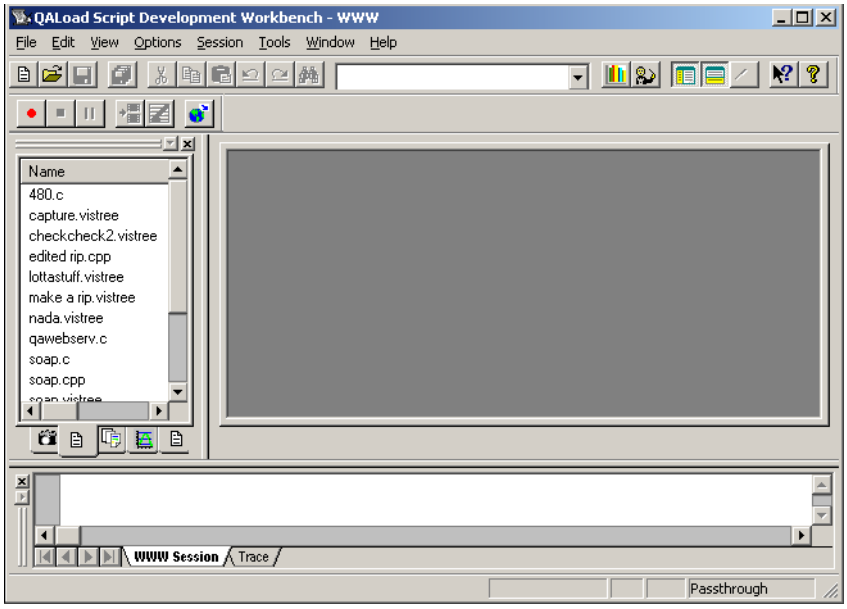
- export a client certificate from the Web browser to *QA*Load and convert it to a format that *QA*Load accepts. See "Exporting Client Certificates from the Browser" on page 2-8 for instructions.
- use *QA*Load to create a client certificate. See "Creating a QALoad Client Certificate" on page 2-11 for instructions.

Exporting Client Certificates from the Browser

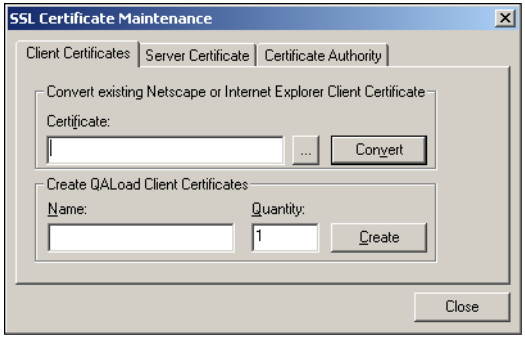
Export a client certificate for *each* virtual user that will run the script. This setup facilitates a one-to-one ratio of client certificates to virtual users, which more realistically simulates your testing environment.

1. Start the Web browser.
2. Locate the client certificate for the Web site you plan to visit.
3. Using your browser's capabilities, export the client certificate (.p12 file) placing the file in a directory where you can access it using the *QA*Load Script Development Workbench.
4. When the browser prompts you to enter a password, *do not* enter a password. If you enter a password, *QA*Load cannot process the file.

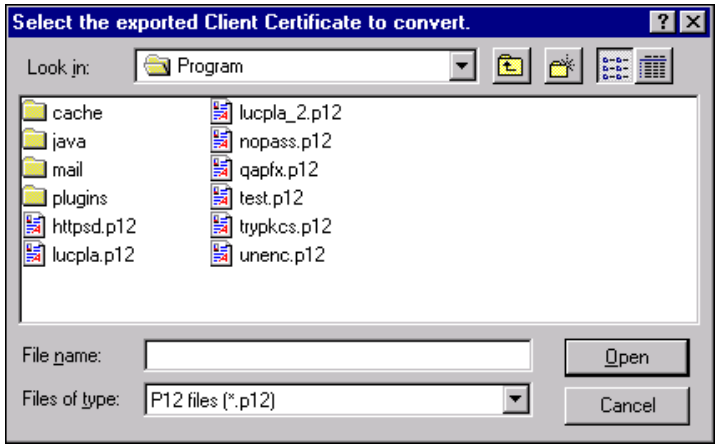
5. Start a WWW Session in the *QA*Load Script Development Workbench by clicking the **WWW Session** button on the toolbar or choosing **WWW** from the **Session** menu.



6. From the **Tools** menu, choose **Maintain Certificates** to open the SSL Certificate Maintenance dialog box.



7. On the **client certificates** tab, click the **Browse** button to browse for the client certificate you exported. The Select the Exported client certificate to Convert dialog box opens.



8. Make sure the Files of Type field specifies P12 files (*.p12).
9. Select the appropriate client certificate and click the **Open** button. The path and file name of the selected client certificate appears in the Enter Certificate to Convert field on the client certificates tab.
10. On the **client certificates** tab, click the **Convert** button to convert the selected client certificate to a format that *QA*Load can recognize.
    - If a message appears indicating a successful convert, click **OK** to close the message and go to Step 11.
    - If a message appears indicating the client certificate did not convert successfully, one of the following scenarios is likely:
      — The client certificate you exported from the browser is not in the correct .p12 format.
      — When you exported the client certificate, you entered a password (see Step 4).

      Click **OK** to close the message. Then, repeat this procedure, making the necessary corrections.
11. Click the **Close** button to exit the SSL Certificate Maintenance dialog box.

After you export and convert a client certificate, you can record SSL requests from a site that requires a client certificate. For instructions, see Chapter 3, “Recording a Test Script”.

Creating a *QA*Load Client Certificate

If you are running a load test with a WWW script containing SSL requests, you should create or export a *QA*Load client certificate for each virtual user that runs the script. This setup facilitates a one-to-one ratio of client certificates to virtual users, which more realistically simulates your testing environment.

To use *QA*Load client certificates, the *QA*Load Certificate Authority and Server Certificate must be valid. If the Certificate Authority or Server Certificate expires, you must create a new one. Refer to the "How Do I?" topics in the *QA*Load Script Development Workbench's online help for instructions on how to create the Certificate Authority or Server Certificate.

1. In the *QA*Load Script Development Workbench, open a WWW Session.
2. From the **Tools** menu, choose **Maintain Certificates** to open the SSL Certificate Maintenance dialog box.
3. On the **Client Certificates** tab in the **Create QALoad Client Certificates** area, type a name for the *QA*Load client certificate in the **Name** field.
4. Type a one (**1**) in the **Quantity** field to create one *QA*Load client certificate. If you need to create additional client certificates before running a load test, you can increase this value.
5. Click the **Create** button to create the *QA*Load client certificate. *QA*Load stores it in the QAload\workbench directory.

After you create a *QA*Load client certificate, you must configure your Web server to accept *QA*Load as the Certificate Authority. Refer to your Web server for more information. After you configure your Web server, you can capture SSL requests from a site that requires a client certificate. For instructions, see Chapter 3, "Recording a Test Script".

Creating a *QA*Load Server Certificate

1. In the *QA*Load Script Development Workbench, 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, named qakey.pem, with the expiration date shown on the Server Certificate tab. The file is automatically saved in the directory Compuware\QALoad\Workbench.
5. Install the newly created server certificate on your server.

Executing SSL Scripts that use Client Certificates

If you are executing SSL scripts that use client certificates, you must manually copy the client certificates in use to the Player machine(s) executing the script(s).

Manually copy the client certificates from the *QA*Load default directory **\Program Files\Compuware\QALoad** to the same default directory on the Player machine.

Tuxedo: Setting Environment Variables

Before you can successfully record from a Tuxedo-based application, you need to set the following environment variables. If you aren't sure how to set or edit an environment variable on your operating system, refer to your operating system help or documentation.

Enter these values carefully. If *QA*Load cannot find the `TUXDIR` environment variable, the environment variable contains an invalid value, or *QA*Load cannot find `%TUXDIR%\bin` in the `PATH` in your environment space, you will receive an error message. Follow the instructions in the error message to correct the appropriate issue. If you do not, *QA*Load may not be able to convert, compile, or play back your script at the appropriate point in your test

1. Define an environment variable called `TUXDIR` that points to the directory in which the Tuxedo client software is installed. For example, `c:\Tuxedo`.
2. Add the following to your `PATH` environment variable: `%TUXDIR%\bin`. For example, `PATH=C:\;C:\Windows;%TUXDIR%\bin`.

SAP: Preparing to Record a Script

Before you can successfully record transactions from an SAP-based application, you must select the Dialog (modal) option under Help>Settings>F4 Help tab in the SAP application. You must do this for the user you are recording. This option must also be selected for playback of SAP scripts.

Chapter 3. Recording a Test Script

This chapter contains the following topics:

- **Overview —** Describes, briefly, the process for recording a script using the *QA*Load Script Development Workbench.
- **Recording a Script —** Details how to record a test script using the *QA*Load Script Development Workbench.
- **Where to Go Next —** Provides information to help you determine what your next step is after recording a test script.

Overview

You create a test script in *QA*Load by recording transactions from the application under test using the *QA*Load Script Development Workbench. The *QA*Load Script Development Workbench contains all the components you need to record a transaction from your application, convert the transaction to a script (C-, C++-, or Java-based, depending on the middleware) that *QA*Load can play back, and compile the script using your own compiler. Essentially, the *QA*Load Script Development Workbench "listens in" on the conversations between your application and the database, Web, or application server. It records those exchanges into a file called the capture file (.cap). The *QA*Load Script Development Workbench then converts the exchanges in your capture file into a C-, C++-, or Java-based script — languages you are probably familiar with — that you can easily modify and compile.

Whatever your middleware, the *QA*Load Script Development Workbench produces test scripts that are consistent from run to run, and are easy to read and modify. *QA*Load's own command set includes equivalents of many functions common to your middleware.

Successful creation of a test script requires four steps:

1. Setting configuration options.
2. Recording a transaction.

3. Converting the recorded transaction to a C-, C++-, or Java-based script, depending on the middleware that was recorded.
4. Compiling the script.

*QA*Load provides configuration options to automate some of these steps. For example, the *QA*Load Script Development Workbench includes options to automatically convert your capture files to scripts and then automatically compile the scripts. You may have already set these options if you completed the procedure "Configuring the QALoad Script Development Workbench" on page 2-2.

If not, Compuware recommends that you set those options now, before you begin recording a script. If you choose not to, instructions for converting and compiling manually can be found in the *QA*Load Script Development Workbench online help.

Recording a Script

Use the following procedure to record a test script using the *QA*Load Script Development Workbench.


Caution

While you are recording, timing information is saved for every call recorded. For any recorded time interval greater than one second, QALoad inserts a sleep (DO_SLEEP) in the script for that number of seconds.
Take caution not to accidentally introduce unwanted sleeps while recording. For instance, if you leave your workstation while recording and return 30 minutes later, *QA*Load will generate a DO_SLEEP command for 30 minutes.

1. Open an appropriate middleware session in the *QA*Load Script Development Workbench.
2. If you haven't already set recording options, select **Options>Record** to open the Record Options wizard. Options available for your middleware session are displayed. For a description of each option, press F1 or click the Help button to access online help for the wizard.

   Set any appropriate recording options and click **OK**.

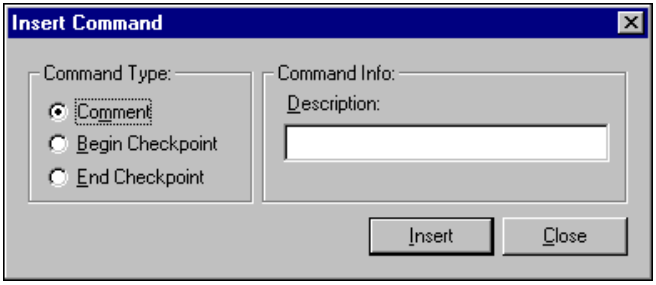

3. From the toolbar, click the **Start Record** button. *QA*Load launches your application and any proxies, if necessary, and begins recording any calls.
4. Run the desired user operations using your application.
5. (WWW only) If you are capturing SSL requests using EasyScript for Secure WWW, the browser generates one or more prompts indicating the following:
   - It does not recognize the authority who signed the server certificate.

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

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

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

   a. Choose **Session>Insert>Command** to open the Insert Command dialog box.



   b. Select the command you want to enter into your capture file. If you select the default, Comment, type your comment in the Description field (for example, login).

   c. Click the **Insert** button.

   d. When you finish, click the **Close** button to close the Insert Command dialog box.

7. When you have recorded your complete transaction, stop the application from which you are recording.

8. After stopping your application, click the **Stop Record** button or choose **Session>Record>Stop**. You will be prompted to save your capture file. By default, capture files (.cap) are saved in the directory QALoad\Middlewares\<*middleware_name*>\Captures.

   If you previously configured the *QA*Load Script Development Workbench to automatically convert and compile your capture file, as recommended in "Configuring the QALoad Script Development Workbench" on page 2-2, your capture file will be converted to a script and then compiled. The results will display in the Output pane, Session tab.

9. When you are finished, see "Where to Go Next" on page 3-5 to determine your next step.


Note

(IIOP, Tuxedo, Winsock, and SAP only) If *QA*Load fails to record from your application using the method outlined in this procedure, use the procedure "Recording Using Manual Application Startup" on page 3-4.

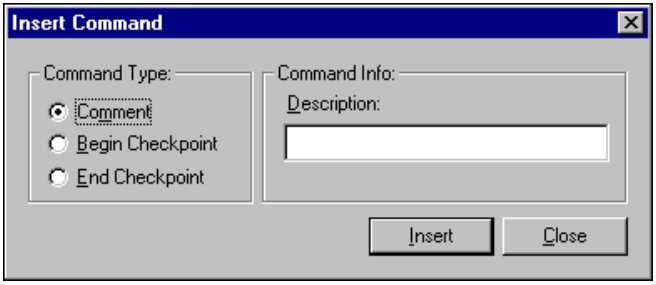
Recording Using Manual Application Startup

(IIOP, Tuxedo, Winsock, and SAP only) Use the following procedure *only* if *QA*Load fails to record from your application using the recommended method detailed in "Recording a Script" on page 3-2.

The following procedure assumes you have already set recording options. If you have not, see "Setting Recording Options" on page 2-5 before beginning this procedure.

1. With the appropriate session open in the *QA*Load Script Development Workbench, select **Options>Record** to open the Record Options wizard.
2. In the **Program Startup** area, select the option **User Started**. Click **OK**.

3. Click the **Start Record** button on the toolbar or select **Session>Record>Start** from the menu.
4. Start your application or (SAP only) start QALSAP.EXE, which is located in the \QALoad directory.
5. (SAP only) Connect to your application server.
6. Run the desired user operations. As you execute them, *QA*Load records your activities in a capture file.
7. (Optional) At any time during the recording process, you can insert any necessary commands or comments into the capture file using the following procedure:
   a. Choose **Session>Insert>Command** to open the Insert Command dialog box.



b. Select the command you want to enter into your capture file. If you select the default, Comment, type your comment in the Description field (for example, login).

c. Click the **Insert** button. When you finish, click the **Close** button to close the Insert Command dialog box.

8. When you finish recording the desired operations, click the **Stop Record** button on the toolbar or select **Session>Record>Stop** from the menu. *QA*Load prompts you to save your capture file. By default, capture files are saved in the directory QALoad\Middlewares\<*middleware_name*>\Captures.

   If you previously configured the *QA*Load Script Development Workbench to automatically convert and compile your capture file, as recommended in "Configuring the QALoad Script Development Workbench" on page 2-2, your capture file will be converted to a script and then compiled. The results will display in the Output pane, Session tab.

9. When you are finished, see "Where to Go Next" on page 3-5 to determine your next step.

Where to Go Next

At this point, you should have a reusable *QA*Load test script containing a complete user transaction. Your script development process should have created the following files:

Table 3-1. *QA*Load files created during the script development process.

File	Description
<*script_name*>.cap	A *QA*Load capture file by the name you designated. Capture files are saved in the directory `\QALoad\Middlewares \<middleware_name> \Captures`.
<*script_name*>.c <*script_name*>.cpp or <*script_name*>.java	A C-, C++-, or Java-based test script by the name you designated. Script files are saved in the directory `\QALoad\Middlewares \<middleware_name> \Scripts`.
<*script_name*>.dll (C-/C++-based) or <*script_name*>.class (Java-based)	An executable version of the script file used in load tests. This file will automatically be downloaded to the appropriate Players at test time. C- and C++-based files are saved in the directory `\QALoad\Scripts`. Java-based script files are saved in the directory `\QALoad\Classes`.

Table 3-1. *QA*Load files created during the script development process.

File	Description
<filename>.rfd *<filename>*.vistree *<filename>*.VisHtml *<filename>*.VisXml	(WWW scripts only) Files related to Visual Scripting, if you converted your script as a Visual Script. For more details, refer to the *QA*Load Script Development Workbench online help.

Capture files and scripts are listed in the *QA*Load Script Development Workbench's Workspace pane, in the Captures and Scripts tabs as shown in Figure 3-1 on page 3-6.

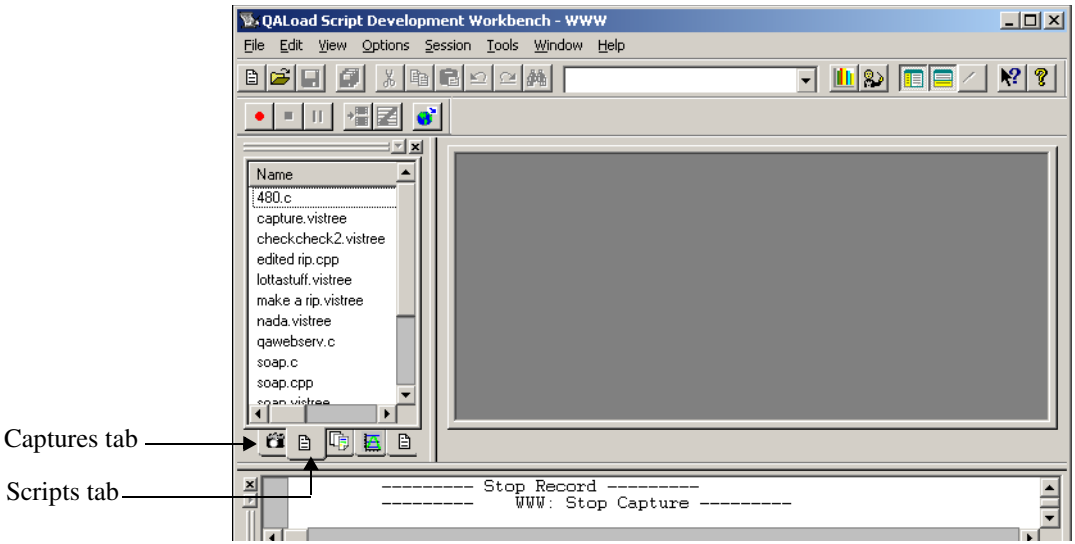


Figure 3-1. Capture files and scripts are listed on the Captures and Scripts tabs.

Look for your capture file and script file on the appropriate tabs. You have several options:

- If you cannot locate your script on the Scripts tab, chances are your capture file was not converted to a script automatically. Press F1 to access the *QA*Load Script Development Workbench online help, and follow the instructions in the "How To..." topic titled "Convert capture files into scripts".
- If your converted script was not compiled automatically during the recording process, you need to compile it manually. Press F1 to access the *QA*Load Script Development Workbench online help, and follow the instructions in the "How To..." topic titled "Compile a script".
- If you would like to learn more about a *QA*Load test script, see Chapter 5, "Understanding a QALoad Test Script".

- If you need to modify your test script to account for special circumstances, such as variable data or authentication, see “Part 3: Customizing a Test Script”.
- If you are ready to set up a test, refer to the *QALoad Testing User’s Guide*.

Part 3:
Customizing a Test Script

Chapter 4. General Advanced Scripting Techniques

After you convert your capture file into a script, you may want to modify it to achieve various performance testing goals. This chapter describes the following scripting techniques to assist you in modifying the script:

- **Defining Transaction Loops** — Provides information about defining transaction loops in the script.
- **Defining Checkpoints** — Provides information about inserting checkpoints into the script to collect timings.
- **Simulating User-Entered Data** — Describes how to modify the script to use variable data from a datapool.

Note

If you converted a WWW script as a Visual Script, the procedures in this chapter do not apply to you. Details about moving the transaction loop, inserting script items, and using datapools and variables with a Visual Script are provided in the *QA*Load online help.

Defining Transaction Loops

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

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 will include checkpoints. Otherwise, you will need to manually insert checkpoints in your script(s) to collect timings.

Simulating User-Entered Data

When you create a script, you will probably have some constant data embedded in the script that automatically enters your application's input fields while recording (for example, an employee number). 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 *QA*Load Script Development Workbench to create, maintain, and use datapool files (.dat) to insert variable data into your scripts.

A datapool can be defined as either central or local:

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

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

Creating a Datapool File

To create a datapool file using the *QA*Load Script Development Workbench:

1. Open a middleware session in the *QA*Load Script Development Workbench.
2. From the **File** menu, select **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, select **File>Save As** to name your datapool file.
9. Click **OK** to save the file. *QA*Load saves the file in your \QALoad\Datapools directory.

Modifying a Datapool File

Complete the following steps to modify a datapool file from the *QA*Load Script Development Workbench.

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 in “Creating a Datapool File” on page 4-2.

Assigning a Central Datapool File

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

5. When you are done, click **OK**.

Using Data Records from a Central Datapool File

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

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

Using Local Datapool Files in a Script

You assign a local datapool file to a specific script by selecting the datapool file and setting any appropriate options using the *QA*Load Script Development Workbench. Each script can use up to sixty-four 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 in “Creating a Datapool File” on page 4-2.

Assigning a Local Datapool

1. Open a session in the *QA*Load Script Development Workbench.
2. In the **Workspace Pane**, click the **Scripts** tab.
3. On the **Scripts** tab, double-click on the appropriate script name to open it in the Workbook Pane.
4. From the **Session** menu, select **Insert>Datapool**. The Insert Datapool Commands dialog box appears.
5. On the Insert Datapool Commands dialog box, click the **Add** button. The Select Datapool dialog box opens.
6. In the **Type** field, select **Local**.

   Note that you can also choose to insert a central datapool from this dialog box. If you choose to insert a central datapool from here, the *QA*Load Script Development Workbench places the Conductor command GET_DATA into the script just after the

BEGIN_TRANSACTION command, bookmarks the command in the margin of the script, and uses any options set for the specified datapool in the *QA*Load Conductor.

7. In the **ID** field, give the datapool a unique identifier. The name can contain alphanumeric characters only. Use underscores (_) for spaces. This ID will help you identify the datapool in your script, for example "ACCOUNT_NUMS".
8. In the **Filename** field, type (or browse for) the fully qualified path of your datapool file. For example:

   c:\Program Files\Compuware\QALoad\Workbench\<middleware_name>\Scripts\datapool.dat
9. If you wish to re-use the datapool records when the script reaches the end of the file, select **Rewind at End of File**. To only use each record once, and then discard it, deselect this option.
10. When you are finished, click **OK**. The selected datapool is displayed on the Insert New Datapool dialog box.
11. Click **OK**.

    The *QA*Load Script Development Workbench will place a `#define` statement identifying the datapool file near the beginning of your script, and place the datapool commands OPEN_DATA_POOL, READ_DATA_RECORD, and CLOSE_DATA_POOL at the default locations in the script. These statements will be bookmarked in the margin for easy identification.
12. When you are finished modifying the script, save any changes.

For detailed information about any of these commands, refer to the Language Reference section of the *QA*Load online help.

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 *QA*Load 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*QA*Load Script Development Workbench to insert your datapool.

For detailed information about any of these commands, refer to the Language Reference section of the *QA*Load online help.

Inserting Variable Data with ActiveData Substitution

The *QA*Load 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 *QA*Load script. This facility also lets you select or edit datapool entries more dynamically, making script development easier and more efficient. Use the procedure that follows to substitute a datapool value or a variable in place of a selected string in your script.

1. Start the appropriate session in the *QA*Load Script Development Workbench.
2. In the Workspace Pane, click the Scripts tab.
3. On the Script tab, double-click the script you wish to open. The script opens in the Workbook Pane.
4. In the script, highlight the string you wish to replace.
5. Right-click anywhere in the highlighted string.
   - To substitute a value from a datapool:
     - Select **ActiveData>Datapool Substitution** from the popup 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 *QA*Load 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 of the *QA*Load online help for detailed information about any of those commands.

- To substitute a variable:
  - Select **ActiveData>Variable Substitution** from the popup menu that opens. The ActiveData Variable Substitution dialog box opens.
  - Assign a variable name for the selected string in the **Variable Name** field.
  - Click **OK**. The *QA*Load 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.

Chapter 5. Advanced Scripting Techniques for WWW

After you convert your capture file into a script, you may want to modify it to achieve various performance testing goals. This chapter describes the following scripting techniques to assist you in modifying the script:

- **Simulating Variable IP Addresses** — Describes how to modify the script to use different source IP addresses from a datapool.
- **Handling Error Messages from the Web Server** — Describes how to handle error messages from the Web server.
- **Simulating CGI Requests** — Describes how *QA*Load simulates CGI Get requests, CGI Post requests, and CGI forms.
- **Simulating JavaScript** — Describes how *QA*Load simulates JavaScript.
- **Executing a Visual Basic Script** — Describes how *QA*Load simulates a Visual Basic script.
- **Executing a Java Applet** — Describes how *QA*Load simulates Java Applets.
- **Simulating Frames** — Describes how *QA*Load simulates frames.
- **Simulating Cookies** — Describes how *QA*Load simulates cookies.
- **Simulating Browser Caching** — Describes how *QA*Load simulates browser caching.
- **Requesting Password-Protected Directories** — Describes how *QA*Load simulates password-protected directories.
- **Using the WWW Convert Options Dialog Box** — Describes and provides a script example for each conversion option.

Simulating Variable IP Addresses

While *QA*Load 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 *QA*Load Player machines with more than one static IP address, *QA*Load 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 *QA*Load Player machine. When you enable IP spoofing in the *QA*Load Conductor, the *Q*ALoad Conductor instructs each *Q*ALoad Player to create the appropriate datapool file at run time. The *Q*ALoad Player will utilize these addresses for connections to HTTP and SSL servers. Each virtual user will receive one address for use with all its connections. If there are more virtual users than addresses, IP addresses will be re-used starting from the beginning of the datapool file.

Modifying a Script to Use Variable IP Addresses

*QA*Load 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 *QA*Load 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 *QA*Load 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. For more information about the IP Spoofing option, see "IP Spoofing" on page 5-91.

Creating a Datapool of IP Addresses

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

1. From the *Q*ALoad program group, open the *Q*ALoad Conductor.
2. Click **Tools>Options**. The Options dialog box appears.
3. Click the **Machines** tab.
4. In the **General Options** area, select **Generate IP Spoofing Data**.

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


Note

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

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

Handling Error Messages from the Web Server

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

Handling Error Messages with Response Codes

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

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

Example

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

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

if((error = DO_GetLastHttpError()) > 399)
{
sprintf(errorString, "Error in response: %d\n", error);
WWW_FATAL_ERROR("Request-host", errorString);
}
```

Handling Error Messages Returned in an HTML Page

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

Using DO_VerifyDocTitle to Verify Page Requests

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

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

Example

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

Searching Response Text for Error Messages

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

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

Example

```
int response;

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

if (strstr (response, "200 OK") == NULL)
WWW_FATAL_ERROR("host", "Response did not have 200 OK");
```

Simulating CGI Requests

This section describes CGI parameter encoding, CGI Get requests, CGI Post requests, and CGI forms.

CGI Parameter Encoding

CGI (Common Gateway Interface) is widely used on World Wide Web sites to provide the ability to run server-side scripts that can take variable input from a Web browser. *QA*Load recognizes when the browser has communicated to a CGI site and will automatically create 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, *QA*Load 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. *QA*Load does not automatically store these types of parameters in variables, but instead provides an easy way to modify the content of the parameters that are being sent in the CGI request via the DO_SetValue command. For more information about the DO_SetValue command, refer to the *QA*Load Language Reference section of the *QA*Load online help.

When you modify parameters that are passed into a *QA*Load CGI request, ensure that all CGI parameters that contain characters that are not alphanumeric (a-Z, 0-9) are encoded prior to being sent to the server. CGI encoding entails inserting the ASCII value of a character, prefixed with the "%" character, into the parameter. *QA*Load 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.

*QA*Load provides an automatic way of performing this encoding via the DO_EncodeString command. For more information about the DO_EncodeString command, refer to the Language Reference section of the *QA*Load online help.

Get Requests

Get requests are handled by the following process.

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

Example Web Page

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

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

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

```
<HTML>
<HEAD>
<TITLE>QALoad WWW Capture Examples</TITLE>
</HEAD>

<BODY>
<A HREF="/cgi-bin/perl_1.pl?name=FRANK">Dynamic HTML Page</
A>
</BODY>
</HTML>
```

Example Script

*QA*Load automatically generates all constructs that are necessary for a CGI Get request.

The following script uses a DO_Http call for the CGI Get request.

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

```
char *Anchor[1];
```

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

DO_InitHttp(s_info);

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

/*
 * Anchor 'http://www.host.com/cgi-bin/perl_1.pl?name=FRANK'
 * 'Dynamic HTML Page'
 */
DO_GetAnchorHREF("Dynamic HTML Page", &Anchor[0]);
DO_SetValue("Anchor000", Anchor[0]);

DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Perl Example Page", TITLE);
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
END_TRANSACTION();
```

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

*QA*Load 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

CGI forms are handled by the following process.

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

Example Web Page

The following Web page contains a CGI form with:

- An action statement
- Input fields
- Hidden fields

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

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

</BODY>
</HTML>
```

Example Script

*QA*Load automatically generates all the constructs that are necessary to make a CGI form request.

The example script features the following:

- 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();
```

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

*QA*Load 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 *QA*Load supports are cookies, title, and the images array. The only image property that *QA*Load supports is src.

Evaluation Errors

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

Example Web Page

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

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

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

function scrollit_r2l(seed)
{
var m1  = "  Welcome to Compuware's QALoad homepeage.";
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: *QA*Load 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();
```

Executing a Visual Basic Script

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

Executing a Java Applet

Java applets are handled by the following process.

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

Example Web Page

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

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

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


<applet code="LScrollText.class" width="500" height="20" >
<PARAM NAME="MESSAGE" VALUE="Scrolling Text created by Java
Applet... >>Click here to Download<<  Use it FREE">
<PARAM NAME="FONTHEIGHT" VALUE="14">
<PARAM NAME="SPEED" VALUE="2">
<PARAM NAME="PIXELS" VALUE="1">
<PARAM NAME="FONTCOLOR" VALUE="0000FF">
<PARAM NAME="BACKCOLOR" VALUE="FFFF00">
<PARAM NAME="TARGET" VALUE="lscrolltext.zip">
</applet>
<br><br><br>
A scrolling message, with custom colors, font size, speed, and
target URL.<br>
The source (.ZIP) file can be downloaded by clicking the
associated area in text window.
<br><br><br><hr>


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

```
</APPLET>


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


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

Example Script

*QA*Load 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: *QA*Load 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 *QA*Load and appear as main requests in the script.

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


DO_VerifyDocTitle("Java Example", TITLE);


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


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

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

Simulating Frames

Frames are handled by the following process.

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

Example Web Page

The following Web page contains four frames.

```
<HTML>
<HEAD>
   <TITLE>FRAME Example</TITLE>
</HEAD>
<! -- Here is the FRAME information for browsers with frames
-->
<FRAMESET Rows="*,*"><!-- Two rows, each equal height -->
   <FRAMESET Cols="*,*"><!-- Two columns, equal width -->
      <FRAME Src="findex.htm"  Name="ul-frame">
      <FRAME Src="findex.htm"  Name="ur-frame">
   </FRAMESET>
   <FRAMESET Cols="*,*"><!-- Two columns, equal width -->
      <FRAME Src="findex.htm" Name="ll-frame">
      <FRAME Src="findex.htm" Name="lr-frame">
   </FRAMESET>
</FRAMESET>

</HTML>
```

Example Script

*QA*Load 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 *QA*Load.

```
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 Cookies

This section describes how *QA*Load 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 *QA*Load 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();
```

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 *QA*Load Script Development Workbench disables browser caching while recording, which means a page is always retrieved. For more information about disabled browser caching, see the Script Development Workbench section of the *QA*Load online help.

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

*QA*Load 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

*QA*Load also handles Windows Domain Authentication (NTLM).

The example script features the following elements:

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

```
BEGIN_TRANSACTION();
DO_NTLMAuthorization("dom1\\frank",
                     "~encr~557A2549474E57444A");
...
...
DO_Http("GET http://www.host.com/ntlm_controlled/"
        "secure.htm HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Successful Test of a NTLM Page", TITLE);
...
...
END_TRANSACTION();
```

Using the WWW Convert Options Dialog Box

This section provides script examples and tips for each conversion option that is available on the WWW Convert Options dialog box.

WWW Convert Options Dialog Box

The WWW Convert Options dialog box contains settings for WWW conversions. It can be accessed by clicking **Options>Convert** while a WWW session is open.

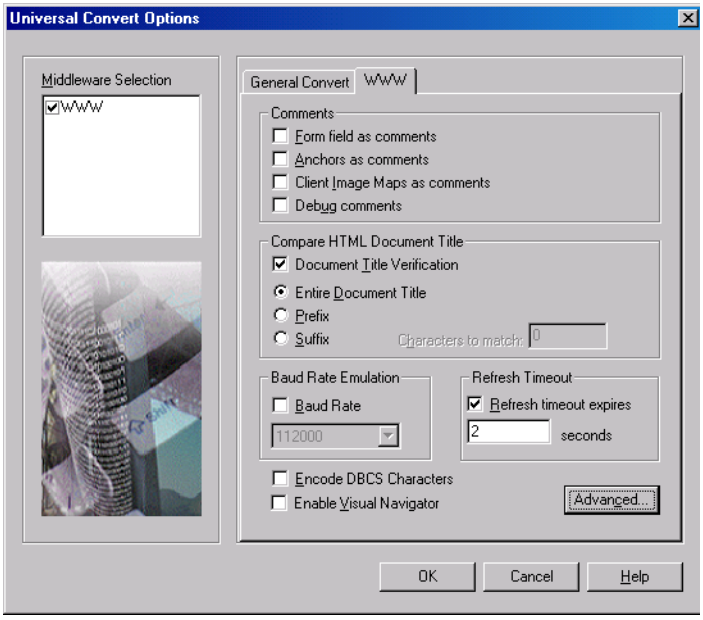


Figure 5-1. WWW Convert Options Dialog Box

Form field as comments

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

Example Web Page

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

```
</HEAD>

<BODY BGCOLOR="#7093DB">
<H2 ALIGN=center>Example of HTTP Forms</H2>
<BR>
<FORM ACTION="/cgi-bin/perl_9.pl" method=post>
<TABLE>
<TR>
<TD>Name:
<TD><INPUT NAME="name" SIZE="20" MAXLENGTH=20>
<TR>
<TD>Password:
<TD><INPUT TYPE =password NAME="password" SIZE="20"
MAXLENGTH=20>
<TR>
<TD>E-Mail Address:
<TD><INPUT NAME= "e-mail" SIZE = "40" MAXLENGTH=80>
<TR>
<TD>Address:
<TD><INPUT TYPE = text NAME = "Address" SIZE = "40"
MAXLENGTH=40>
<TR>
<TD>City:
<TD><INPUT NAME="city" SIZE="40" MAXLENGTH=40>
<TD ALIGN=left>State:
<TD ALIGN=left><INPUT NAME="state" SIZE="2" MAXLENGTH=2
ALIGN=left>
<TD ALIGN=left>Zip:
<TD ALIGN=left><INPUT NAME="zip" SIZE="5" MAXLENGTH=5>
<TR>
<TD VALIGN=top>Favorite Color:
<TD><SELECT NAME="options">
<OPTION>Red
<OPTION>Orange
<OPTION>Yellow
<OPTION>Green
<OPTION selected=on>Blue
<OPTION>Indigo</OPTION>
<OPTION>Violet</OPTION>
```

```
</SELECT>
<TR>
<TR>
<TD VALIGN=top>Color of your money:
<TD><SELECT NAME="dates" multiple="multiple">
<OPTION selected=on>Red
<OPTION>Blue
<OPTION>Green</OPTION>
<OPTION>Beige</OPTION>
</SELECT>
<TR>
<TD VALIGN=top>Comments:
<TD><TEXTAREA NAME="comments" COLS=40 ROWS=5></TEXTAREA>
</TABLE>

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

<P>
<TABLE>
<TR>
<TD VALIGN=top>Testing:
<TD><INPUT TYPE=radio CHECKED NAME="test"
VALUE="capture">Capture<BR>
    <INPUT TYPE=radio NAME="test" VALUE="replay">Replay<BR>
    <INPUT TYPE=radio NAME="test"
VALUE="loadtest">Loadtest<BR>
<TR>
<TR>
<TD>Web page to append to reply:
<TD><INPUT TYPE=file NAME="web page">
</TABLE>
<BR>
There is a hidden field containing data here: <INPUT
TYPE=hidden NAME="hidden" VALUE="This rocks!">
Here is another hidden field: <INPUT TYPE=hidden
NAME="hidden1" VALUE="Web testing is fun">
<BR>
<BR>
```

```
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="colors.gif" width="200"
height="100">
</TABLE>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="eye.gif" width="80" height="60">
</TABLE>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="devplatform.gif" width="48"
height="43">
</TABLE>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="enterprise_sm.gif" width="42"
height="41">
</TABLE>
<BR>
<TABLE ALIGN=center>
<TR>
<TD><INPUT TYPE=submit NAME="submit">
<TD><INPUT TYPE=reset>
</TABLE>
</FORM>

</BODY>
</HTML>
```

Form field as comments – Yes

The following example has the **Form field as comments** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : Yes
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
```

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

for(i=0;i<1;i++)
ActionURL[i]=NULL;
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/forms.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("Forms Example", TITLE);

/* ActionURL[0]="http://www.host.com/cgi-bin/perl_9.pl" */
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
/* Form:1 text Name: name, Value: */
/* Form:1 text Name: password, Value:  */
/* Form:1 text Name: e-mail, Value:  */
/* Form:1 text Name: Address, Value: */
/* Form:1 text Name: city, Value: */
/* Form:1 text Name: state, Value: */
/* Form:1 text Name: zip, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: options, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 select Name: dates, Value: */
/* Form:1 text Name: comments, Value: */
/* Form:1 checkbox Name: echo, Value: */
/* Form:1 radio Name: test, Value: capture, */
/* Form:1 radio Name: test, Value: replay, */
```

```
/* Form:1 radio Name: test, Value: loadtest, */
/* Form:1 *unknown* Name: web page, Value: , */
/* Form:1 hidden Name: hidden, Value: This rocks!, */
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1,
                      &Field[0]);
/* Form:1 hidden Name: hidden1, Value: Web testing is fun */
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1,
                      &Field[1]);
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 *unknown* Name: , Value: */
/* Form:1 submit Name: submit, Value: */
/* Form:1 *unknown* Name: , Value: */


/* Request: 2  From: Forms Example */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");

DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
       "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
      "{name}&{password}&{e-mail}&{Address}&{city}&{state}&"
```

```
        "{zip}&{options}&{dates}&{comments}&{echo}&{test}&"
        "{web+page}&{hidden}&{hidden1}&{submit}");

...
...
```

Form field as comments – No

The following example has the **Form field as comments** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
```

```
 */
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
...
...
for(i=0;i<2;i++)
Field[i]=NULL;

for(i=0;i<1;i++)
ActionURL[i]=NULL;
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/forms.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("Forms Example", TITLE);

DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1,
                      &Field[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1,
                      &Field[1]);

/* Request: 2  From: Forms Example */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
```

```
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");

DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
       "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
       "{name}&{password}&{e-mail}&{Address}&{city}&{state}"
        "&{zip}&{options}&{dates}&{comments}&{echo}&{test}"
        "&{web+page}&{hidden}&{hidden1}&{submit}");
...
...
for(i=0; i<2; i++)
{
free(Field[i]);
Field[i]=NULL;
}

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

Anchors as comments

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

Example Web Page

```
<HTML>
<HEAD>
<TITLE>QALoad WWW Capture Examples</TITLE>
</HEAD>
```

```
<BODY>
<HR>
<IMG SRC="../logo.gif" ALIGN=LEFT width="127" height="129">
<IMG SRC="../logo.gif" ALIGN=RIGHT width="127" height="129">

<BR><BR>
<CENTER><H2><EM>QALoad WWW Capture Examples</EM></H2>
<A HREF=../default.htm><h3>Return to welcome homepage.</h3></
A>
<BR></CENTER><HR>

<CENTER>
<TABLE CELLSPACING="10">

<TR>
<TH ALIGN=left>LINK:
<TH ALIGN=left>DESCRIPTION:

<TR>
<TD><A HREF="../standard.htm">Standard HTML Homepage</A></
TD>
<TD>Static page w/ images (.GIF 87a, 89a), sound files (.WAV),
and assorted links</TD>

<TR>
<TD><a href="../subs.htm">Multiple Inline Images Page</a></
TD>
<TD>Static page with 16 inline images</TD>

</TABLE>

</CENTER>

</BODY>
</HTML>
```

Anchors as comments – Yes

The following example has the **Anchors as comments** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : Yes
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
/* Declare Variables */
int i;
char *Anchor[1];
...
```

```
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("QALoad WWW Capture Examples", TITLE);
/* Anchors 'http://www.host.com/default.htm' 'Return to
welcome homepage.' */
/* Anchors 'http://www.host.com/standard.htm' 'Standard HTML
Homepage' */
DO_GetAnchorHREF( "Standard HTML Homepage", &Anchor[0]);
/* Anchors 'http://www.host.com/subs.htm' 'Multiple Inline
Images Page' */

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

/* Variable: Anchor000 links to: Standard HTML Homepage on
   page: QALoad WWW Capture Examples */
DO_SetValue("Anchor000", Anchor[0]);

DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("\"Standard HTML Example\"", TITLE);
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
```

Anchors as comments – No

The following example has the **Anchors as comments** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
/* Declare Variables */
int i;
char *Anchor[1];
...
```

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

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

DO_VerifyDocTitle("QALoad WWW Capture Examples", TITLE);
DO_GetAnchorHREF( "Standard HTML Homepage", &Anchor[0]);

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

/* Variable: Anchor000 links to: Standard HTML Homepage on
   page: QALoad WWW Capture Examples */
DO_SetValue("Anchor000", Anchor[0]);

DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}
```

Client Image Maps as Comments

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

Example Web Page

```
<html><head></head><body>
<center><h2>Client-side version of clickable imagemap</h2>
```

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

<MAP NAME="title">
<AREA SHAPE="rect" COORDS="1,108,115,124" HREF="sup.htm"></
AREA>
<AREA SHAPE="rect" COORDS="119,107,234,124"
HREF="reg.htm"></AREA>
<AREA SHAPE="rect" COORDS="235,107,352,124"
HREF="fea.htm"></AREA>
<AREA SHAPE="rect" COORDS="353,107,466,124"
HREF="tech.htm"></AREA>
<AREA SHAPE="rect" COORDS="0,127,156,144"
HREF="htmlsam.htm"></AREA>
<AREA SHAPE="rect" COORDS="157,127,312,144"
HREF="exampro.htm"></AREA>
<AREA SHAPE="rect" COORDS="313,127,466,144"
HREF="feedback.htm"></AREA>
<AREA SHAPE="rect" COORDS="0,0,466,143"
HREF="invalid.htm"></AREA>
</MAP>

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

Client Image Maps as Comments – Yes

The following example has the **Client Image Maps as Comments** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                    : 80 characters
 *  Sleep Seconds                 : 1
 *  Auto Checkpoints              : Yes
 * WWW
 *  Form Field Comments           : No
 *  Anchors as Comments           : No
```

```
 *  Client Maps as Comments          : Yes
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
for(i=0;i<1;i++)
ClientMapURL[i]=NULL;
...
...
/* Request: 1  */

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

/* Client Map:1 Region:1 HREF: http://www.host.com/sup.htm */
/* Client Map:1 Region:2 HREF: http://www.host.com/reg.htm */
DO_GetClientMapHREF(MAP(1), REGION(2), &ClientMapURL[0]);
/* Client Map:1 Region:3 HREF: http://www.host.com/fea.htm */
/* Client Map:1 Region:4 HREF: http://www.host.com/tech.htm
*/
```

```
/* Client Map:1 Region:5 HREF: http://www.host.com/
   htmlsam.htm */
/* Client Map:1 Region:6 HREF: http://www.host.com/
   exampro.htm */
/* Client Map:1 Region:7 HREF: http://www.host.com/
   feedback.htm*/
/* Client Map:1 Region:8 HREF: http://www.host.com/
   invalid.htm */

/* Request: 2  */
DO_SetValue("ClientMap000", ClientMapURL[0]);

DO_Http("GET {*ClientMap000} HTTP/1.0\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(ClientMapURL[i]);
ClientMapURL[i]=NULL;
}

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

Client Image Maps as Comments – No

The following example has the **Client Image Maps as Comments** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
```

```
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
for(i=0;i<1;i++)
ClientMapURL[i]=NULL;
...
...
/* Request: 1  */

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

DO_GetClientMapHREF(MAP(1), REGION(2), &ClientMapURL[0]);

/* Request: 2  */
DO_SetValue("ClientMap000", ClientMapURL[0]);

DO_Http("GET {*ClientMap000} HTTP/1.0\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
```

```
free(ClientMapURL[i]);
ClientMapURL[i]=NULL;
}

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

Debug comments

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

Debug comments – Yes

The following example has the **Debug Comments** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : Yes
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
```

```
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");

/*  Received reply: <QALoad WWW Capture Examples> */
DO_VerifyDocTitle("QALoad WWW Capture Examples", TITLE);
...
...
```

Debug Comments – No

The following example has the **Debug Comments** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
```

```
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");


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

Document Title Verification

There are three supported methods for verifying a document title, which are shown in the following sections. Document title verification can be a good tool for detecting and handling error messages that are returned in an HTML page.

Example Web Page

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


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

Document Title Verification - Yes

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

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
```

```
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
...
...
```

Prefix (Characters to match - 5)

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Prefix
 *    Characters To Match           : 5
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
 */
...
...
```

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

DO_VerifyDocTitle("Welco", PREFIX);

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

Suffix (Characters to match - 4)

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Suffix
 *    Characters To Match           : 4
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
```

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

DO_VerifyDocTitle("Page", SUFFIX);

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

Document Title Verification – No

The following example has the **Document Title Verification** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : No
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
```

```
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

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

Baud Rate

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

The DO_SetBaudRate command is inserted in the script with the specified baud rate as its only parameter. If baud rate emulation must be asymmetric (upload rate is different than the download rate), use the DO_SetBaudRateEx command. DO_SetBaudRateEx takes two parameters: the upload baud rate and the download baud rate. For more information about the DO_SetBaudRateEx command, refer to the Language Reference section of the *QA*Load online help.

Baud Rate (57600) – Yes

The following example has the **Baud Rate** option selected and set to 57600.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
```

```
 *  Baud Rate Emulation              : Yes
 *    Baud Rate                      : 57600
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_SetTransactionStart();
DO_SetBaudRate(57600);
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();

...
...
```

Baud Rate Emulation – No

The following example has the **Baud Rate Emulation** option cleared.

```
/* Converted using the following options:
 * General:
```

```
 *  Line Split                        : 80 characters
 *  Sleep Seconds                     : 1
 *  Auto Checkpoints                  : Yes
 * WWW
 *  Form Field Comments               : No
 *  Anchors as Comments               : No
 *  Client Maps as Comments           : No
 *  Debug Comments                    : No
 *  Doc Title Verification            : Yes
 *   Compare By                       : Entire Document Title
 *  Baud Rate Emulation               : No
 *  Encode DBCS Characters            : No
 *  Cache                             : No
 *  Dynamic Redirect                  : Yes
 *  Dynamic Cookies                   : Yes
 *  Process Subrequests               : Yes
 *  Persistent Connections            : Yes
 *  Reuse SSL Session ID              : Yes
 *  Max Concurrent Connection         : 4
 *  Max Connection Retries            : 4
 *  Server Response Timeout           : 120
 *  HTTP Version Detection            : Auto
 *  ActiveData                        : Yes
 *  IPSpoofing                        : No
 *  Streaming Media                   : No
 *  Hostnames as IP Addresses         : No
 *  Strip All Cookies From Requests   : No
 */
...
...

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

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

Refresh Timeout

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

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

Example Web Page

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

Refresh Timeout – Yes

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

```
/* Converted using the following options:
...
...
 *  Baud Rate Emulation              : No
 *  Enable Refresh Timeout           : Yes
 *   Refresh Timeout                 : 10
 *  Encode DBCS Characters           : No
...
```

```
...

      DO_SetTransactionStart();
      DO_SetRefreshTimeout(10);
      DO_SetMaxBrowserThreads(2);

...
...

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

Refresh Timeout – No

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

```
/* Converted using the following options:
...
 *  Baud Rate Emulation              : No
 *  Enable Refresh                   : No
 *  Encode DBCS Characters           : No
...
...

      BEGIN_TRANSACTION();

      DO_SetTransactionStart();
      DO_SetMaxBrowserThreads(2);

...
...

/* Request: 1  */
DO_Http("GET http://host/path/to/page.pl HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Just Wait", TITLE);
```

```
DO_SLEEP(5);

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

Encode DBCS Characters

When this option is selected, double-byte characters are converted into octal format. This must be enabled for a capture with DBCS characters, so that the double-byte characters can be viewed in a legible format.

Example Web Page

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

<body onload="document.search.p.focus();" topmargin=8>
...
...
</body>
</html>
```

Encode DBCS Characters – Yes

The following example has the **Encode DBCS Characters** option selected.

```
/* Converted using the following options:
```

```
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters  : Yes
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...

/* Request: 1  */
DO_Http("GET http://kr.yahoo.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("\276\337\310\304!"
                  "\304\332\270\256\276\306", TITLE);
...
...
```

Encode DBCS Characters – No

The following example has the **Encode DBCS Characters** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters  : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...
/* Request: 1  */
DO_Http("GET http://kr.yahoo.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("야후! 코리아", TITLE);
```

. . .
. . .

Enable Visual Navigator

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

Enable Visual Navigator – Yes

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

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

Enable Visual Navigator – No

If you do not select the Enable Visual Navigator option, *QA*Load generates a standard C script. All normal and advanced conversion options apply to the script.

WWW Advanced Convert Options Dialog Box

The WWW Advanced dialog box contains advanced options for WWW conversions and is accessed by the **Advanced** button on the WWW Convert Options dialog box.

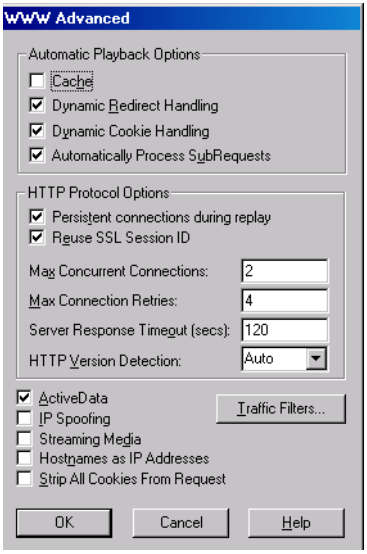


Figure 5-2. WWW Advanced Convert Options Dialog Box

Cache

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

Cache – Yes

The following example has the **Cache** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                    : 80 characters
 *  Sleep Seconds                 : 1
 *  Auto Checkpoints              : Yes
 * WWW
 *  Form Field Comments           : No
 *  Anchors as Comments           : No
 *  Client Maps as Comments       : No
 *  Debug Comments                : No
 *  Doc Title Verification        : Yes
 *   Compare By                   : Entire Document Title
 *  Baud Rate Emulation           : No
 *  Encode DBCS Characters        : No
 *  Cache                         : Yes
```

```
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_Cache(TRUE);      /* Enable cache */
...
...
END_TRANSACTION();
...
...
```

Cache – No

The following example has the **Cache** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
```

```
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_Cache(FALSE);      /* Disable cache */
...
...
END_TRANSACTION();
...
...
```

Dynamic Redirect Handling

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

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

Dynamic Redirect Handling – Yes

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

The following example has the **Dynamic Redirect Handling** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : Yes
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...
/* Declare Variables */
```

```
int i;
char *Anchor[1];
...
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_DynamicRedirectHandling(TRUE);
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("QALoad WWW Capture Examples", TITLE);
...
...
DO_GetAnchorHREF( "Redirected Webpage", &Anchor[0]);
...
...
DO_SetValue("Anchor000", Anchor[0]);

DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Successful Test of Dynamic Redirect"
                  "Sample",
                  TITLE);
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
Anchor[i]=NULL;
}

END_TRANSACTION();
```

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

Dynamic Redirect Handling – No

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

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : Yes
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : No
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
```

```
/* Declare Variables */
int i;
char *Anchor[1];
...
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
DO_DynamicRedirectHandling(FALSE);
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");
DO_VerifyDocTitle("QALoad WWW Capture Examples", TITLE);

DO_GetAnchorHREF( "Redirected Webpage", &Anchor[0]);

DO_SetValue("Anchor000", Anchor[0]);

DO_Http("GET {*Anchor000} HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Document Moved", TITLE);

DO_Http("GET http://www.host.com/redir/frm.pl HTTP/"
        "1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of Dynamic Redirect"
                  "Sample",
                  TITLE);

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

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

Dynamic Cookie Handling

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

Example Web Page

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>
```

Dynamic Cookie Handling – Yes

The following example has the **Dynamic Cookie Handling** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
```

```
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : Yes
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_DynamicCookieHandling(TRUE);
...
...
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n\r\n");

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

Dynamic Cookie Handling – No

The following example has the **Dynamic Cookie Handling** option cleared.

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

```
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : No
 *  Process Subrequests             : No
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
 */
...
...
char *Cookie[4];
...
...
for(i=0;i<2;i++)
Cookie[i]=NULL;
...
...
BEGIN_TRANSACTION();
DO_DynamicCookieHandling(FALSE);
...
```

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

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

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

/* Request: 2  */
DO_SetValue("cookie000", Cookie[0]);  /* NUM=1 */
DO_SetValue("cookie001", Cookie[1]);  /* SQUARE=1 */

DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n"
        "Cookie: {*cookie000}; {*cookie001}\r\n\r\n");
...
...
DO_HttpCleanup();
for(i=0; i<2; i++)
{
free(Cookie[i]);
Cookie[i]=NULL;
}

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

Automatically Process SubRequests

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

Example Web Page

```
<html>

<head>
```

```
<title>Page Of Subs</title>
</head>

<body>

<p>The page of subrequests</p>
<p>.<img border="0" src="win2000.gif" width="456"
height="124"><img border="0" src="APACHE.GIF" width="259"
height="32"><img border="0" src="banner.gif" width="470"
height="64"><img border="0" src="COLORS.GIF" width="200"
height="100"></p>

</body>

</html>
```

Automatically Process SubRequests – Yes

The following example has the **Automatically Process SubRequests** option selected.

```
/*
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
```

```
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_AutomaticSubRequests(TRUE);
...
...
DO_Http("GET http://www.host.com/subs.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("Page Of Subs", TITLE);
...
...
END_TRANSACTION();
...
...
```

Automatically Process SubRequests - No

The following example has the **Automatically Process SubRequests** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
```

```
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : No
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
 */

...
...
BEGIN_TRANSACTION();
...
...
DO_AutomaticSubRequests(FALSE);
...
...
DO_Http("GET http://www.host.com/subs.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("Page Of Subs", TITLE);

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

/* Request: 3  From: Page Of Subs */
```

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

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

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

Persistent Connections During Replay

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

Persistent Connections During Replay – Yes

The following example has the **Persistent connections during replay** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
```

```
 *  Cache                              : No
 *  Dynamic Redirect                   : Yes
 *  Dynamic Cookies                    : Yes
 *  Process Subrequests                : Yes
 *  Persistent Connections             : Yes
 *  Reuse SSL Session ID               : Yes
 *  Max Concurrent Connection          : 4
 *  Max Connection Retries             : 4
 *  Server Response Timeout            : 120
 *  HTTP Version Detection             : Auto
 *  ActiveData                         : Yes
 *  IPSpoofing                         : No
 *  Streaming Media                    : No
 *  Hostnames as IP Addresses          : No
 *  Strip All Cookies From Requests    : No
 */

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

Persistent Connections During Replay – No

The following example has the **Persistent connections during replay** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                         : 80 characters
 *  Sleep Seconds                      : 1
 *  Auto Checkpoints                   : Yes
 * WWW
 *  Form Field Comments                : No
 *  Anchors as Comments                : No
 *  Client Maps as Comments            : No
```

```
 *  Debug Comments                    : No
 *  Doc Title Verification            : Yes
 *   Compare By                       : Entire Document Title
 *  Baud Rate Emulation               : No
 *  Encode DBCS Characters            : No
 *  Cache                             : No
 *  Dynamic Redirect                  : Yes
 *  Dynamic Cookies                   : Yes
 *  Process Subrequests               : Yes
 *  Persistent Connections            : No
 *  Reuse SSL Session ID              : Yes
 *  Max Concurrent Connection         : 4
 *  Max Connection Retries            : 4
 *  Server Response Timeout           : 120
 *  HTTP Version Detection            : Auto
 *  ActiveData                        : Yes
 *  IPSpoofing                        : No
 *  Streaming Media                   : No
 *  Hostnames as IP Addresses         : No
 *  Strip All Cookies From Requests   : No
 */

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

Reuse SSL Session ID

This option is available only on an SSL installation of *QA*Load. 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.

Reuse SSL Session ID – Yes

The following example has the **Reuse SSL session ID** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */

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

Reuse SSL Session ID – No

The following example has the **Reuse SSL session ID** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
```

```
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
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();
...
...
```

Max Concurrent Connections

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

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

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */

...
...
```

```
BEGIN_TRANSACTION();

DO_SetTransactionStart();
DO_SetMaxBrowserThreads(4);
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
END_TRANSACTION();
...
...
```

Max Connection Retries

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

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
```

```
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 4
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
...
...
DO_SetMaximumRetries(4);
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
```

Server Response Timeout

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

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
```

```
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : Yes
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_InitHttp(s_info);
DO_SetTimeout(120); /* Maximum time to wait for HTTP Reply */
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Welcome to QAWEBSERV", TITLE);
...
...
```

HTTP Version Detection

A WWW script can be set to 1.1, 1.0, or Auto. When set to 1.1, all requests and subrequests are sent as HTTP/1.1. When set to 1.0, all HTTP requests and subrequests are sent as HTTP/1.0. When set to Auto, the individual DO_Http and DO_Https commands determine the version of HTTP to use.

The default setting for this option is Auto, which is the best option for most scripts. However, if your application requires HTTP version 1.1 or there are other special scripting conditions, the script can be set to use a specific version of HTTP.

HTTP Version Detection: Auto/1.1/1.0

The following example has the **HTTP Version Detection** field set to Auto.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
```

```
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
DO_HTTPVersion("Auto");
...
...
END_TRANSACTION();
...
...
```

ActiveData

When this option is enabled, the generated script will automatically be variablized. Otherwise, all requests are not modified after the capture.

The following example shows a capture of a form, how it is variablized by the ActiveData option, and not variablized when the ActiveData option is cleared.

Example Web Page

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

<BODY BGCOLOR="#7093DB">
<H2 ALIGN=center>Example of HTTP Forms</H2>
<BR>
<FORM ACTION="/cgi-bin/perl_9.pl" method=post>
<TABLE>
<TR>
<TD>Name:
<TD><INPUT NAME="name" SIZE="20" MAXLENGTH=20>
<TR>
<TD>Password:
<TD><INPUT TYPE =password NAME="password" SIZE="20"
MAXLENGTH=20>
```

```
<TR>
<TD>E-Mail Address:
<TD><INPUT NAME= "e-mail" SIZE = "40" MAXLENGTH=80>
<TR>
<TD>Address:
<TD><INPUT TYPE = text NAME = "Address" SIZE = "40"
MAXLENGTH=40>
<TR>
<TD>City:
<TD><INPUT NAME="city" SIZE="40" MAXLENGTH=40>
<TD ALIGN=left>State:
<TD ALIGN=left><INPUT NAME="state" SIZE="2" MAXLENGTH=2
ALIGN=left>
<TD ALIGN=left>Zip:
<TD ALIGN=left><INPUT NAME="zip" SIZE="5" MAXLENGTH=5>
<TR>
<TD VALIGN=top>Favorite Color:
<TD><SELECT NAME="options">
<OPTION>Red
<OPTION>Orange
<OPTION>Yellow
<OPTION>Green
<OPTION selected=on>Blue
<OPTION>Indigo</OPTION>
<OPTION>Violet</OPTION>
</SELECT>
<TR>
<TR>
<TD VALIGN=top>Color of your money:
<TD><SELECT NAME="dates" multiple="multiple">
<OPTION selected=on>Red
<OPTION>Blue
<OPTION>Green</OPTION>
<OPTION>Beige</OPTION>
</SELECT>
<TR>
<TD VALIGN=top>Comments:
<TD><TEXTAREA NAME="comments" COLS=40 ROWS=5></TEXTAREA>
</TABLE>
```

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

<P>
<TABLE>
<TR>
<TD VALIGN=top>Testing:
<TD><INPUT TYPE=radio CHECKED NAME="test"
VALUE="capture">Capture<BR>
    <INPUT TYPE=radio NAME="test" VALUE="replay">Replay<BR>
    <INPUT TYPE=radio NAME="test"
VALUE="loadtest">Loadtest<BR>
<TR>
<TR>
<TD>Web page to append to reply:
<TD><INPUT TYPE=file NAME="web page">
</TABLE>
<BR>
There is a hidden field containing data here: <INPUT
TYPE=hidden NAME="hidden" VALUE="This rocks!">
Here is another hidden field: <INPUT TYPE=hidden
NAME="hidden1" VALUE="Web testing is fun">
<BR>
<BR>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="colors.gif" width="200"
height="100">
</TABLE>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="eye.gif" width="80" height="60">
</TABLE>
<TABLE ALIGN=center>
```

```
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="devplatform.gif" width="48"
height="43">
</TABLE>
<TABLE ALIGN=center>
<TR>
<TD ALIGN=center>Don't Click This
<TR>
<TD><INPUT TYPE=image SRC="enterprise_sm.gif" width="42"
height="41">
</TABLE>
<BR>
<TABLE ALIGN=center>
<TR>
<TD><INPUT TYPE=submit NAME="submit">
<TD><INPUT TYPE=reset>
</TABLE>
</FORM>

</BODY>
</HTML>
```

ActiveData – Yes

When this option is enabled, variablizations are done in the script based on the settings of other conversion options that become available. The following example has the **ActiveData** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
```

```
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
 */
...
...
/* Declare Variables */
int i;
char *Field[2];
char *ActionURL[1];
...
...
for(i=0;i<2;i++)
Field[i]=NULL;

for(i=0;i<1;i++)
ActionURL[i]=NULL;
...
...
/* Request: 1  */
DO_Http("GET http://www.host.com/forms.htm HTTP/"
        "1.0\r\n\r\n");
```

```
DO_VerifyDocTitle("Forms Example", TITLE);
DO_GetFormActionStatement(FORM(1), &ActionURL[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden", 1,
                      &Field[0]);
DO_GetFormValueByName(FORM(1), "hidden", "hidden1", 1,
                      &Field[1]);

/* Request: 2  From: Forms Example */
DO_SetValue("action_statement0", ActionURL[0]);
DO_SetValue("name", "joe");
DO_SetValue("password", "");
DO_SetValue("e-mail", "");
DO_SetValue("Address", "");
DO_SetValue("city", "");
DO_SetValue("state", "");
DO_SetValue("zip", "");
DO_SetValue("options", "Blue");
DO_SetValue("dates", "Red");
DO_SetValue("comments", "");
DO_SetValue("echo", "on");
DO_SetValue("test", "capture");
DO_SetValue("web+page", "");
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
DO_SetValue("submit", "Submit Query");

DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
       "Content-Type: application/x-www-form-urlencoded\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
      "{name}&{password}&{e-mail}&{Address}&{city}&{state}&"
        "{zip}&{options}&{dates}&{comments}&{echo}&{test}&"
        "{web+page}&{hidden}&{hidden1}&{submit}");
...
...
for(i=0; i<2; i++)
{
free(Field[i]);
Field[i]=NULL;
}
```

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

ActiveData – No

When this option is not selected, variablizations are not done in the script regardless of the other options. The following example has the **ActiveData** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 5
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : No
 *  IPSpoofing                      : No
```

```
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/forms.htm HTTP/"
        "1.0\r\n\r\n");

DO_VerifyDocTitle("Forms Example", TITLE);

DO_Http("POST http://www.host.com/cgi-bin/perl_9.pl HTTP/"
     "1.0\r\n"
     "Content-Type: application/x-www-form-urlencoded\r\n"
     "Content-Length: {*content-length}\r\n\r\n"
    "name=joe&password=&e-mail=&Address=&city=&state=&zip=&"
    "options=Blue&dates=Red&comments=&echo=on&test=capture&"
     "web+page=&hidden=This+rocks%21&"
     "hidden1=Web+testing+is+fun&submit=Submit+Query");

DO_VerifyDocTitle("Forms Example - Results", TITLE);
...
...
END_TRANSACTION();
...
...
```

IP Spoofing

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

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

For more information about the DO_IPSpoofEnable command, refer to the Language Reference section of the *QA*Load online help.

IP Spoofing – Yes

The following example has the **IP Spoofing** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : Yes
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests : No
 */

...
...

DO_InitHttp(s_info);
DO_IPSpoofEnable("");
```

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

IP Spoofing – No

The following example has the **IP Spoofing** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 4
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : Yes
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
```

```
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
...
...
END_TRANSACTION();
...
...
```

Streaming Media

*QA*Load supports two types of streaming media:

- RealOne Player
- Windows Media Player

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

Note

Streaming media is not supported through firewalls and across proxies.

Advanced RealOne Player Media Options

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

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

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

Enable client-side performance statistics for RealNetworks Streaming Media.

```
void DisableStatisticsRP(void);
```

Disable client side performance statistics for RealNetworks Streaming Media.<|

Streaming Media – Yes

The following example has the **Streaming Media** option selected and uses RealOne Player.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 5
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : No
 *  IPSpoofing                      : No
 *  Streaming Media                 : Yes
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...
#include "do_www.h"
```

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

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

DownloadMediaRP("http://rm.host.com:8099/ramgen/demo.rm",
                0);
...
...
END_TRANSACTION();
...
...
```

Windows Media Player Example

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
```

```
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : No
 *  IPSpoofing                       : No
 *  Streaming Media                  : Yes
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
#include "do_www.h"
#include "SMPLayer.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/"
        "1.0\r\n\r\n");

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

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

DownloadMediaFromASX(0);
...
...
END_TRANSACTION();
...
...
```

Streaming Media – No

The following example has the **Streaming Media** option cleared and uses RealOne Player.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 5
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : No
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
 *  Strip All Cookies From Requests  : No
 */
...
...
#include "do_www.h"
...
...
```

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

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

DO_Http("GET http://rm.host.com:8099/ramgen/demo.rm "
        "HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Windows Media Player Example

The following example has the **Streaming Media** option cleared and uses Windows Media Player.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
```

```
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : No
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
#include "do_www.h"
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/index.htm HTTP/
        "1.0\r\n\r\n");

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


DO_Http("GET http://www.host.com/wmp-test.asx HTTP/
        "1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Hostnames as IP Addresses

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

Hostnames as IP Addresses – Yes

The following example has the **Hostnames as IP Addresses** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 5
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : No
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : Yes
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://172.22.24.39/ HTTP/1.0\r\n\r\n");
```

```
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
...
...
END_TRANSACTION();
...
...
```

Hostnames as IP Addresses – No

The following example has the **Hostnames as IP Addresses** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
 *  Process Subrequests             : Yes
 *  Persistent Connections          : Yes
 *  Reuse SSL Session ID            : No
 *  Max Concurrent Connection       : 4
 *  Max Connection Retries          : 5
 *  Server Response Timeout         : 120
 *  HTTP Version Detection          : Auto
 *  ActiveData                      : No
 *  IPSpoofing                      : No
 *  Streaming Media                 : No
 *  Hostnames as IP Addresses       : No
```

```
 *  Strip All Cookies From Requests  : No
 */
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");

DO_VerifyDocTitle("Welcome to The Main Page", TITLE);
...
...
END_TRANSACTION();
...
...
```

Strip All Cookies From Request

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

Strip All Cookies From Request – Yes

The following example has the **Strip All Cookies From Request** option selected.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
```

```
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : No
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : Yes
 */
...
...
DO_Http("GET http://qawebserv.compuware.com/cgi-bin/
        "cookies5.pl HTTP/1.0\r\n\r\n");

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

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

Strip All Cookies From Request – No

The following example has the **Strip All Cookies From Request** option cleared.

```
/* Converted using the following options:
 * General:
 *  Line Split                       : 80 characters
 *  Sleep Seconds                    : 1
 *  Auto Checkpoints                 : Yes
 * WWW
 *  Form Field Comments              : No
 *  Anchors as Comments              : No
 *  Client Maps as Comments          : No
 *  Debug Comments                   : No
```

```
 *  Doc Title Verification           : Yes
 *   Compare By                      : Entire Document Title
 *  Baud Rate Emulation              : No
 *  Encode DBCS Characters           : No
 *  Cache                            : No
 *  Dynamic Redirect                 : Yes
 *  Dynamic Cookies                  : Yes
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : No
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n"
        "Cookie: username=username\r\n\r\n");
...
...
```

Traffic Filters Dialog Box

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

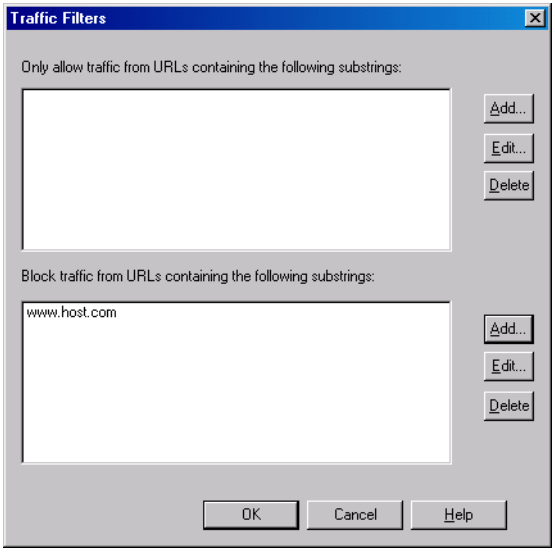


Figure 5-3. Traffic Filters Dialog Box

Filter Requests – Yes

In the following example, a traffic filter has been set to exclude requests with URLs that contain the string "www.host.com". Compare the following script to the script at "Filter Requests – No" on page 5-107.

```
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 * WWW
 *  Form Field Comments             : No
 *  Anchors as Comments             : No
 *  Client Maps as Comments         : No
 *  Debug Comments                  : No
 *  Doc Title Verification          : Yes
 *   Compare By                     : Entire Document Title
 *  Baud Rate Emulation             : No
 *  Encode DBCS Characters          : No
 *  Cache                           : No
 *  Dynamic Redirect                : Yes
 *  Dynamic Cookies                 : Yes
```

```
 *  Process Subrequests              : Yes
 *  Persistent Connections           : Yes
 *  Reuse SSL Session ID             : No
 *  Max Concurrent Connection        : 4
 *  Max Connection Retries           : 5
 *  Server Response Timeout          : 120
 *  HTTP Version Detection           : Auto
 *  ActiveData                       : No
 *  IPSpoofing                       : No
 *  Streaming Media                  : No
 *  Hostnames as IP Addresses        : No
 *  Strip All Cookies From Requests  : No
 */
...
...
DO_InitHttp(s_info);
...
...
/* Exclude requests with URLs containing */
DO_BlockRequestsFrom("www.host.com;");
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://rm.host.com:8099/ramgen/realmp3.mp3 "
        "HTTP/1.0\r\n"
        "Referer: http://www.host.com/index.htm\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Filter Requests – No

The following example does not use traffic filters.

```
/* Converted using the following options:
 * General:
```

```
 *  Line Split                        : 80 characters
 *  Sleep Seconds                     : 1
 *  Auto Checkpoints                  : Yes
 * WWW
 *  Form Field Comments               : No
 *  Anchors as Comments               : No
 *  Client Maps as Comments           : No
 *  Debug Comments                    : No
 *  Doc Title Verification            : Yes
 *   Compare By                       : Entire Document Title
 *  Baud Rate Emulation               : No
 *  Encode DBCS Characters            : No
 *  Cache                             : No
 *  Dynamic Redirect                  : Yes
 *  Dynamic Cookies                   : Yes
 *  Process Subrequests               : Yes
 *  Persistent Connections            : Yes
 *  Reuse SSL Session ID              : No
 *  Max Concurrent Connection         : 4
 *  Max Connection Retries            : 5
 *  Server Response Timeout           : 120
 *  HTTP Version Detection            : Auto
 *  ActiveData                        : No
 *  IPSpoofing                        : No
 *  Streaming Media                   : No
 *  Hostnames as IP Addresses         : No
 *  Strip All Cookies From Requests   : No
 */
...
...
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
```

```
DO_VerifyDocTitle("Welcome to The Main Page", TITLE);

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

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

DO_Http("GET http://rm.host.com:8099/ramgen/realmp3.mp3 "
        "HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Chapter 6. Advanced Scripting Techniques for Tuxedo

- **Managing Tuxedo Application Data Flow** — Describes how *QA*Load manages Tuxedo buffers, how data passes from one *QA*Load Tuxedo command to another, and how to include data in the script.

Managing Tuxedo Application Data Flow

To get the script to execute properly, you need to ensure that data flows through the script as it would normally flow through your application. The following sections describe how *QA*Load manages Tuxedo buffers, how data passes from one *QA*Load Tuxedo command to another, and how to encode data in a script.

Managing Tuxedo Buffers

Tuxedo clients use typed buffers to transmit data between Tuxedo clients and servers. You can create a typed buffer by using the `tpalloc` command and specifying the buffer type and size. *QA*Load supports the following Tuxedo buffer types:

- FML
- FML32
- STRING
- CARRAY
- X_OCTET
- VIEW
- VIEW32

For example, to allocate a 4096 byte FML buffer on the client, use the following code:

```
char *buffer;
buffer = tpalloc( "FML", "", 4096 );
```

To place data into the buffer, use the following code:

```
FChg( buffer, fieldid, oc, "data", 4 );
```

Where `buffer` is the Tuxedo-allocated (tpalloc) buffer, `fieldid` is the field value, and `oc` is the field occurrence.

To simplify buffer management and provide more comprehensive error checking, *QA*Load Tuxedo scripts automatically handle buffer management. Instead of having to work with buffer pointers, *QA*Load's Tuxedo commands hide the buffer pointers by managing an array of buffers behind the scenes. The commands identify buffers using a mnemonic name such as Buf1, which translates into the array index, rather than a buffer pointer.

The following example shows how a Tuxedo script manages a buffer allocation for the command Do_Tuxtpcall.

```
Do_Tuxtpalloc( Buf1 , "FML", 1024 ); );
Do_TuxFinit( Buf1 );
Do_TuxFMLData( test_carray, 1, "abcdefg" );
Do_TuxFMLData( test_long, 1, "12345" );
Do_Tuxtpcall( "OPEN_TEST1", Buf1 , Buf2 , 0 );
```

In the example above, the `Do_Tuxtpalloc` command allocates a buffer named Buf1. `Do_TuxFinit` clears any previous contents of Buf1. The `Do_TuxFMLData` commands load data into the most recent buffer that `Do_TuxFinit` clears; therefore, the `Do_TuxFMLData` parameter list does not include Buf1.

Following the setup of the buffer, the `Do_Tuxtpcall` makes a service call to `OPEN_TEST1`. The parameter list of the `Do_Tuxtpcall` includes an input and output buffer. In the example above, the input buffer is `Buf1` and the output buffer is `Buf2`. The final parameter of zero indicates that special Tuxedo flags are not specified.

*QA*Load automatically determines if a buffer type is FML or FML32 and calls the appropriate Tuxedo API routines.

Note that a command is not available to free a previously allocated buffer. When the script executes a `Do_Tuxtpalloc` command, *QA*Load checks to see whether the buffer associated with a specified buffer index was previously allocated. If *QA*Load determines that the buffer was previously allocated, it frees the buffer using Tuxedo's tpfree prior to allocating it.

Passing Data Between Tuxedo Commands

When a Tuxedo client application executes, it may pass data from one API call to another. A script that needs to emulate an application needs to pass data in the same way the application passes data. The following example shows how to use *QA*Load commands to pass output data from one `Do_Tuxtpcall` as input to another `Do_Tuxtpcall`.

```
/* Declare Variables for Account ID and encode Account ID */
```

```
char AcctID[16];
char EncAcctID[32];

/* Set up input buffer with Account Name for retrieving
Account ID */
Do_Tuxtpalloc( Buf1 , "FML", 1024 );
Do_Tuxtpalloc( Buf2 , "FML", 1024 );
Do_TuxFinit( Buf1 );
Do_TuxFMLData( ACCT_NAME, 0, "Gerard Plumbing");

/* Retrieve Account ID using the name */
Do_Tuxtpcall( "getAcctIdFromName", Buf1 , Buf2 , 0);

/* Extract the Account id from the output buffer */
Do_TuxgetFMLData( Buf2 , ACCT_ID, 0, AcctID);

/* Account id may be special characters, so encode it */
Do_Tuxencode( EncAcctID, AcctID, strlen(AcctID) );

/* Load up the buffer for the next call */
Do_TuxFinit( Buf1 );
Do_TuxFMLData( ACCT_ID, 0, EncAcctID );

/* Call to get account detail */
Do_Tuxtpcall( "getAcctDetail", Buf1 , Buf2 , 0 );
```

In the example above, the first `Do_Tuxtpcall` retrieves an account ID from the account name. The account name is placed into `Buf1` (input buffer), and the account ID is placed into `Buf2` (output buffer).

The account ID is retreived from `Buf2` using the `Do_TuxgetFMLData` command. The `Do_TuxgetFMLData` command retrieves data from a typed buffer using the Tuxedo field and occurrence identifiers.

When data is returned using the `Do_TuxgetFMLData` command, it is returned in its internal form, without encoding. Yet, the `Do_TuxFMLData` command, which loads data into the Tuxedo buffers, requires that special characters are encoded. Therefore, the

Do_Tuxencode command is used to encode the data before using it as input to the second Do_Tuxtpcall.

Note

If you manipulate an encoded string, remember that all non-printable and some special characters occupy three bytes in the array. Make sure you take this into account during character substitution. Note that the EncAcctID variable, in the example above, is larger than the AcctID variable.

You can also use the Do_TuxgetTuxBuffer command to work with data from a Tuxedo buffer. The Do_TuxgetTuxBuffer command returns the actual address of a Tuxedo buffer given a buffer name. Once you have the pointer to the buffer, you can use native Tuxedo commands such as Fadd, FChg, etc. for FML or memcpy for CARRAY-type data to input data into or retrieve data from a Tuxedo buffer.

VIEW and VIEW32 buffers are accessed using compiler macros automatically generated in *QA*Load's Convert facility. For example, a view called testVw16 is accessed using the macro VW_testVw16(buffer _index) as shown in the sample below.

```
/* Allocate buffer space for testVw16 in buffer #2,   */
/* and set values.  */
Do_Tuxtpalloc( Buf2, "VIEW:testVw16", sizeof(struct testVw16)
    );
VW_testVw16(Buf2)->tv16intneg = -1234;
```

Encoding String Data in Scripts

You may need to include data in the script so it can get placed into a buffer. A technique called string encoding makes non-printable characters readable in the script. Note that you can use encoded strings for data that *QA*Load's Convert facility places in the script or for data you place in the script.

The following *QA*Load commands use encoded strings as parameters:

- Do_TuxFMLData
- Do_Tuxcarray
- Do_Tuxxoctet
- Do_Tuxstring
- Do_Tuxtpinit
- Do_TuxSetViewData
- Do_TuxBuildBuffer
- Do_TuxAppendBuffer

A string is encoded using the following rules:

- all alpha and numeric characters (0-9, a-z, and A-Z) are preserved intact

- all non-alpha numeric characters within the range of ASCII 32 (space) to ASCII 125 (}) are preserved intact, except the following:
  - backslash (\ )
  - ampersand (&)
  - double quote (" ")
  - pipe (|)
- null characters are encoded as a tilde (~)
- all other characters are encoded as a three-byte sequence of an ampersand (&) followed by two lowercase hex digits representing the ASCII value of the character.

The following example illustrates encoding:

Original String: `0  1 2 A B C D a b c - & | (null)`

Encoded String: `0 1 2 A B C D a b c - &26&7c~`

Chapter 7. Advanced Scripting Techniques for Winsock

After you have converted your capture file into a script, you may want to modify it to achieve a particular testing goal. This chapter describes the following scripting techniques to assist you in modifying your script.

- **Understanding Data Representation in the Script** — Describes how *QA*Load represents the data that was captured within the Winsock functions.
- **Handling Winsock Application Data Flow** — Describes how to modify the script to include variables rather than hard-coded values.
- **Modifying *QA*Load's Functions to Incorporate Dynamic Data** — Describes how to substitute octal data in a Winsock script from a datapool file. Also describes how to modify a DO_WSK_Send() command.
- **Saving Server Replies** — Describes how to save a received buffer by using the DO_WSK_Recv() command instead of the DO_WSK_Expect() command.
- **Parsing Server Replies for Values** — Describes how to use the SkipExpr(), ScanExpr(), ScanSkip(), and ScanString() commands to parse a value from a reply.

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, *QA*Load 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.

Table 7-1. Character Encoding Table

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

Table 7-1. Character Encoding Table

Code	Octal	Decimal	Char
\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 hardcoded values returned by the server with variables. The following original and modified code examples demonstrate this technique.

Original Code

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

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


/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 */
#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
   /* Declare Variables */
   SET_ABORT_FUNCTION(abort_function);
```

```
    DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");
    // Checkpoints have been included by the convert process
    DefaultCheckpointsOn();
    DO_WSK_Init(s_info);
    SetTimeout(20);   /* Wait up to 20 seconds for each
    expected pattern */
    SYNCHRONIZE();
    BEGIN_TRANSACTION();
    DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
    DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
    DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);
    ////////////////////////////////////////////////////
    //  The session id returned by the server is
    //  unique to each connection
    ////////////////////////////////////////////////////
    * 21bytes: SessionID=jrt90847\r\n */
    DO_WSK_Expect(S1, "\n");
    ////////////////////////////////////////////////////
    //  This unique id is then used for subsequent
    //  requests
    ////////////////////////////////////////////////////
    /* 34 bytes */
    DO_WSK_Send(S1,
    "SessionID=jrt90847\r\n:^B^@^@^@^B^@^@^@^A^@^@^@");
    /* 15 bytes: ID Accepted#^@\r\n */
    DO_WSK_Expect(S1, "\n");
    DO_WSK_Closesocket(S1);
    END_TRANSACTION();
    REPORT(SUCCESS);
    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User %i:ABORTED.", S_task_id);
    EXIT();
}
```

Modified Code

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

```
/*
 *  wsk-AdvancedTechniques_modified.c
 *
 *  This script contains support for the following
*   middlewares:
 *      - Winsock
 */
/* Converted using the following options:
 * General:
 *  Line Split                      : 80 characters
 *  Sleep Seconds                   : 1
 *  Auto Checkpoints                : Yes
 */
#define SCRIPT_VER 0x00000005UL
#include <stdio.h>
#include "smacro.h"
#include "do_wsk.h"
/* set function to call on abort*/
void abort_function(PLAYER_INFO *s_info);
#ifndef NULL
#define NULL  0
#endif
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
   /* Declare Variables */
   char Buffer[64];
   char SendBuffer[64];
   int    nBytesReceived = 0;
   SET_ABORT_FUNCTION(abort_function);
   DEFINE_TRANS_TYPE("wsk-AdvancedTech_1.c");
   // Checkpoints have been included by the convert process
   DefaultCheckpointsOn();
   DO_WSK_Init(s_info);
   SetTimeout(20);   /* Wait up to 20 seconds for each
   expected pattern */
```

```
    SYNCHRONIZE();
    BEGIN_TRANSACTION();
    DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
    DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
    DO_WSK_Connect(S1, "172.22.24.125", 2100, AF_INET);
    ////////////////////////////////////////////////////
    //  The reply from the server is read into
    //  the Buffer variable.  We will then have
    //  the unique Session ID for this connection.
    //  Also need to null-terminate the buffer
    //  after receiving.
    ////////////////////////////////////////////////////
    DO_WSK_Recv(S1, Buffer, 64, 0, &nBytesReceived);
    Buffer[nBytesRecieved] = '\0';
    /* 21bytes: SessionID=jrt90847\r\n */
    //DO_WSK_Expect(S1, "\n");
    ////////////////////////////////////////////////////
    //  Finally, substitute the Session ID received from
    //  the server with the one coded in the script.
    ////////////////////////////////////////////////////
    sprintf(SendBuffer, "%s:^B^@^@^@^B^@^@^@^A^@^@^@",
    Buffer);
    DO_WSK_Send(S1, SendBuffer);
    /* 34 bytes */
    //DO_WSK_Send(S1,
    "SessionID=jrt90847:^B^@^@^@^B^@^@^@^A^@^@^@");
    /* 15 bytes: ID Accepted#^@\r\n */
    DO_WSK_Expect(S1, "\n");
    DO_WSK_Closesocket(S1);
    END_TRANSACTION();
    REPORT(SUCCESS);
    EXIT();
    return(0);
}
void abort_function(PLAYER_INFO *s_info)
{
    RR__printf("Virtual User %i:ABORTED.", S_task_id);
    EXIT();
}
```

Modifying QALoad's Functions to Incorporate Dynamic Data

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

Scenario 1:

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

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

Example

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

```
/* Declare variables */
char temp[40];
...
BEGIN_TRANSACTION();
GET_DATA();
...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT)
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
strcpy(temp,VARDATA(1));
OctalToChar(temp); //used to convert octal strings
                   //to their binary format
DO_WSK_Send(S1,temp);
//DO_WSK_Send(S1,"\121\101\122\165\156");
DO_WSK_Closesocket(S1);
```

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

Scenario 2:

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

Example

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

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

...
BEGIN_TRANSACTION();

...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1,ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);
//original   DO_WSK_Send(S1,"name=Jim");
strcpy( buffer, "Mark");
sprintf( sendbuffer, "name=%s", buffer);
DO_WSK_Send(S1, sendbuffer);
/* 2 bytes: ok */
DO_WSK_Expect(S1,"ok");
DO_WSK_Closesocket(S1);
```

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

Saving Server Replies

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

Using the Response() and ResponseLength() Commands

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

number of characters that were received. The following example uses the Response() and ResponseLength() commands.

Example

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

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

...
BEGIN_TRANSACTION();

...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);
/* 21 bytes: You are now connected */
DO_WSK_Expect(S1, "d");
// Used to store the data that was received by the
// DO_WSK_Expect
temp = Response();


// Used to get the size of the response that was received
// so far
x = ResponseLength();
/*The line below will print the length of the response and
the actual response*/
RR__printf("length = %d, and response= %s",x, temp);
DO_WSK_Closesocket(S1);
```

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

Using the DO_WSK_Recv() Command

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

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

Example

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

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

...
BEGIN_TRANSACTION();

...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);
/* 21 bytes: You are now connected */
memset(temp,'\0',45);
DO_WSK_Recv(S1,temp,45,0,&size);
RR__printf("size=%d string=%s",size,temp);
DO_WSK_Closesocket(S1);
```

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


Note

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 *QA*Load’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.

Table 7-2. Common UNIX-style regular 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 interanl 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.

Chapter 8. Advanced Scripting Techniques for SQL Server

After you convert your capture file into a script, you may want to modify it to process variable data within SQL Server calls. This chapter describes scripting techniques to assist you in modifying the script accordingly.

Variablizing SQL Server Scripts

This section describes the following techniques for variablizing SQL Server scripts:

Capturing a *select* Value from a Stored Procedure

In SQL Server, the `select` statement can be used to retrieve a value from the stored procedure. These values can be retrieved in a *QA*Load script by adding the `addResultVar()` and `getResultVar()` commands into the script. The `addResultVar()` command will need to be added before the `DO_getResults()` command for the stored procedure call. After the `DO_getResults()` call, a `getResultVar()` will return the value as a string for the parameter named in the `addResultVar()` call.

In the following example, we retrieve a selected number value from a stored procedure as a string and then convert it into an integer.

Example SQL Server Stored Procedure with *select* value

```
create procedure inc_test_sp
(
```

```
@first_param int
)
as
begin
   select second_param = @first_param + 1
end
```

Example *QA*Load Script Code

```
strcpy(sql_statement,     /* >>   1   << */
   "execute inc_test_sp @sample_param ='{01}' ");
DO_substr(sql_statement, 1, "100" );
BEGIN_CHECKPOINT();   /* #1: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
while (DO_dbGetResults(0))
;
END_CHECKPOINT(25);   /* #25: Stored Procedure */
```

Step 1: Add the necessary variable declarations

To declare the following variables at the beginning of the *QA*Load script, locate the following lines in the script:

```
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
```

Then add the bolded lines below after them, as shown:

```
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
char    szSecondParam[20];   /* Assume number length < 20!! */
long     iSecondParam;
```

Step 2: Alter the script code that calls the stored procedure

Locate the code that calls the stored procedure and modify it to extract the return value by adding the bolded lines as shown in the following example.

```
strcpy(sql_statement,     /* >>   1   << */
   "execute inc_test_sp @sample_param ='{01}' ");
DO_substr(sql_statement, 1, "100" );
BEGIN_CHECKPOINT();   /* #1: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
DO_addResultVar( "second_param" ); /* Add THIS line HERE !!!*/
```

```
while (DO_dbGetResults(0))
;
END_CHECKPOINT(25);   /* #1: Stored Procedure */
strcpy( szSecondParam, DO_getResultVar( "second_param" ) );
iOutputReqID = atoi( szSecondParam );
RR__printf("Second Param (string): %s", szSecondParam );
RR__printf("Second Param (int): %d", iSecondParam );
```

Using a Retrieved Value as a Parameter to a Stored Procedure

Often, values retrieved from a stored procedure are used as input parameters to subsequent stored procedures while recording a SQL Server session. You can parameterize the stored procedure calls in your *QA*Load script as shown in the following example.

Original *QA*Load Script Code

```
strcpy(sql_statement,/* >>   2 << */
   "execute use_inc_value_sp @inc_value ={01}");
DO_substr(sql_statement, 1, "101" );
BEGIN_CHECKPOINT();   /* #2: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
while (DO_dbGetResults(0))
;
```

*QA*Load Script Code Modified to Use String Value

```
strcpy(sql_statement,    /* >>   2 << */
   "execute use_inc_value_sp @inc_value ={01}");
/* Note that szSecondParam was declared and received the
   value in the steps in Part 1 */
DO_substr(sql_statement, 1, szSecondParam );
BEGIN_CHECKPOINT();      /* #2: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
while (DO_dbGetResults(0))
;
```

Capturing an OUTPUT Parameter Value from a Stored Procedure Call

In SQL Server, the `OUTPUT` parameter can return a value from a stored procedure. These values can be retrieved in a *QA*Load script by adding the `addResultVar()` and `getRe-`

sultVar() commands into the script. The addResultVar() command will need to be added before the DO_getResults() command for the stored procedure call. After the DO_getResults() call, a getResultVar() will return the value as a string for the parameter named in the addResultVar() call.

In the following example, we retrieve an OUTPUT parameter number value from a stored procedure as a string and then convert it into an integer.

Example SQL Server Stored Procedure with OUTPUT Parameter

```
create procedure output_ret_inc_test_sp
(
@input_param int,
@output_param int OUTPUT
)
as
begin
   select @output_param = @input_param + 1
end
```

Original *QA*Load Script Code

```
strcpy(sql_statement,     /* >>   1   << */
   "execute output_ret_inc_test_sp @input_param ='{01}' ");
DO_substr(sql_statement, 1, "100" );
BEGIN_CHECKPOINT();       /* #1: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
while (DO_dbGetResults(0))
;
END_CHECKPOINT(25);      /* #25: Stored Procedure */
```

Step 1: Add the necessary variable declarations

To declare the following variables at the beginning of the *QA*Load script, locate the following lines in the script:

```
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
```

Then add the bolded lines below after them, as shown:

```
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
char    szOutputParam[20];  /* Assume number length < 20!! */
```

```
long    iOutputParam;
```

Step 2: Alter the script code that calls the stored procedure

Locate the code that calls the stored procedure and modify it to extract the return value by adding the bolded lines as shown in the following example.

```
strcpy(sql_statement,     /* >>   1   << */
   "execute output_ret_inc_test_sp @input_param ='{01}' ");
DO_substr(sql_statement, 1, "100" );
BEGIN_CHECKPOINT();      /* #1: Stored Procedure */
DO_dbcmd(0, sql_statement);
DO_dbsqlexec( 0 );
DO_addResultVar( "@output_param" ); /* Add THIS line HERE! */
while (DO_dbGetResults(0))
;
END_CHECKPOINT(25);      /* #1: Stored Procedure */
strcpy( szOutputParam, DO_getResultVar( "@output_param" ) );
iOutputReqID = atoi( szOutputParam );
RR__printf("Output Param (string): %s", szOutputParam );
RR__printf("Output Param (int): %d", iOutputParam );
```

Chapter 9. Advanced Scripting Techniques for SAP

After you convert your capture file into a script, you may want to modify it to achieve various performance testing goals. This chapter describes the following scripting techniques to assist you in modifying the script:

- **Required Commands** — Describes commands that are required in an SAP script.
- **Error Handling and Reporting** — Describes how the SAP middleware handles error handling and reporting.
- **Handling Multiple Logons** — Describes how to handle multiple user logons.
- **Checking the SAP Status Bar** — Describes how to set up tests that compare against the messages that appear in the SAP status bar.
- **Object Life Span** — Describes the life span of objects in the SAP environment.

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();
```

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");
HRESULT 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");
```

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();
```

Handling Multiple Logons

You may need to modify your script to handle multiple logins 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.

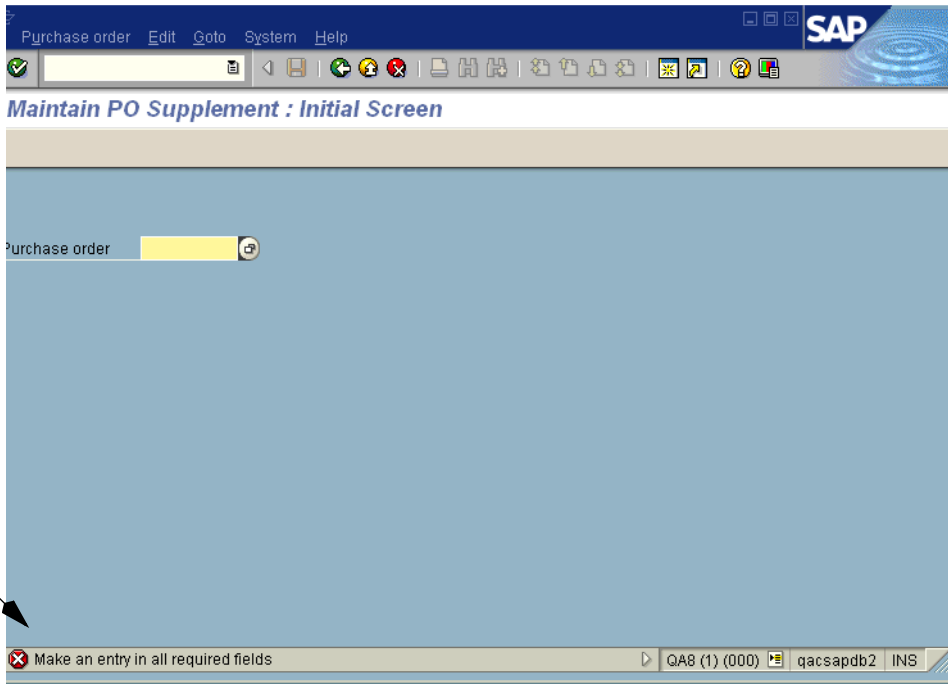
The following example demonstrates the usage of the `SAPGuiPropIdStrExists` and `SAPGuiPropIdStrExistsEnd` commands to handle multiple logons:

```
...
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*".

Chapter 10. Advanced Scripting Techniques for Citrix

After you convert your capture file into a script, you may want to modify it to achieve various performance testing goals. This chapter describes the following scripting techniques to assist you in modifying the script:

- **Handling Dynamic Windows** — Describes how to modify the script to deal with dynamic window creation.
- **Using the WaitForScreenUpdate Command** — Describes how to modify the script to handle mouse click/window synchronization issues.
- **Handling Dynamic Window Titles** — Describes techniques to detect match patterns for windows with dynamic title names, how to create a match string using wildcards and substrings, and how to use the `SetWindowMatchName` command to set the match pattern for the window prior to the `WaitForWindowCreate` command in the script.
- **Handling Dynamic Windows That Require User Action** — Describes how to modify the script to handle windows that intermittently appear but, on occasions where the window appears, requires some user action, such as a mouse click.
- **Moving the Citrix Connect and Disconnect Outside the Transaction Loop** — Describes the steps required during recording and script development to allow for Citrix logon and logoff actions to be moved outside the script transaction loop.
- **Handling Citrix Server Farms** — Describes how to modify the script to connect to a Citrix server farm.

Handling Dynamic Windows

During conversion, `WaitForWindowCreate` 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 `WaitForWindowCreate` commands that may be referencing dynamic windows.

Using the WaitForScreenUpdate 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 `WaitForScreenUpdate` command in the script after the action that causes the window to appear. The parameters for the `WaitForScreenUpdate` 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.

For more information about the `WaitForScreenUpdate` command, refer to the *QA*Load online help.

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 `SetEnableWildcardMatching` 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 `WaitForWindowCreate` 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 `SetWindowMatchName` command prior to the `WaitForWindowCreate` call for the dynamic window.** When adding the `SetWindowMatchName` 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 `WaitForWindowCreate` 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
SetWindowMatchTitle( CWI_13, "Microsoft Word" );
WaitForWindowCreate(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
SetWindowMatchTitle(CWI_13,"SampleClientApp - User: *" );
WaitForWindowCreate(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
SetWindowMatchTitle( CWI_13, "Sample Application: ?" );
WaitForWindowCreate(CWI_13);
```

Handling Dynamic Windows That Require User Action

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

1. Capture a session in which the dynamic window appears and the user performs the action to dismiss the window.

   This may require multiple attempts to capture the window. Once this is captured in a recording, save the script as a temporary script.

2. If the window did not appear in the primary script, extract the code snippet from the temporary script that acts on the dynamic window and insert it into the real script.

   The code usually consists of a `Point` command and a `Click` command for this window. Insert the commands after the `WaitForWindowCreate` command for the dynamic window. In addition, extract and insert the Citrix window information object constructor call and delete call to the relevant parts of the script, changing the object name to avoid conflicting with existing window objects. Ensure that the additional code is not inserted between a `Point` command and a `Click` command in the primary script.

3. Add a special `SetWindowMatchTitle` command immediately before the `WaitForWindowCreate` command.

   The first parameter of the `SetWindowMatchTitle` command should be the correct window object. The second parameter contains a special wildcard match "*" that enables the `Click` command to accept any window title, which ensures that even if the matching window does not appear, the command still executes successfully.

4. If the window appears in the primary script, comment out the `WaitForWindowCreate` command for the dynamic window.

   Because the window itself may not appear, the `WaitForWindowCreate` command should be commented out.

5. Validate the script. If the validation is not successful, ensure that steps 2-4 were performed correctly.

In the following example's scenario, the ICA Seamless Window Agent window does not appear in the primary script, but appears intermittently when the primary script is replayed, causing those replay sessions to fail. A second Citrix script, which includes the

window appearance, is recorded and the `Point` and `Click` commands are extracted from this script and inserted into the primary script, with the window object changed to match the object in the primary script. In addition, the Citrix window object constructor call and delete call are added in the appropriate places in the script, and the `Click` command is changed to refer to this object. In the following example, the text in bold represents code that was manually inserted into the location in the primary script where the window appears in the secondary script.

```
CtxWI *CWI_99 = new CtxWI(0x10052, "ICA Seamless Host Agent",
0, 0, 391, 224);
...
SetWindowMatchTitle( CWI_99, "*" );
Point(190, 203);
Click(CWI_99, 0, L_BUTTON, NONE);
Point(300, 400);
...

delete CWI_99; // "ICA Seamless Host Agent"
```

Moving the Citrix Connect and Disconnect Outside the Transaction Loop

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

Recording

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

1. Insert a comment such as “Logged in to Citrix” after the Citrix logon but before any windows have been opened.
2. Ensure that all application windows are closed before disconnecting from the Citrix session.
3. Insert a command such as “Ready to log off Citrix” before the Citrix logoff sequence is initiated.

Ensure that the first comment is added after the user has logged on and closed all login-related dialog boxes, but before any applications are started. Similarly, the second comment must be placed after all applications have been closed, but before the user logs off.

Scripting

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

Handling Citrix Server Farms

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

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

```
.
.
.
/* 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 */
```

```
SetConnectTimeout(90);
SetDisconnectTimeout(90);
SetWindowTimeout(30);
SetPingTimeout(20);
SetWaitPointTimeout(30);
SetWindowVerification(TRUE);
SetDomainLoginInfo(CitrixUsername, CitrixPassword, Citrix-
Domain);
SetICAFile("PRD desktop.ica");
SetEnableCounters(TRUE);
SetWindowRetries(5, 5000);
SetEnableWildcardMatching(TRUE);

SYNCHRONIZE();
```

Index

T

V

W