

QALoad 5.02

Online Help

Customer Support Hotline:
1-800-538-7822

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

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

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

© 1998-2004 Compuware Corporation. All rights reserved. Unpublished - rights reserved under the Copyright Laws of the United States.

U.S. GOVERNMENT RIGHTS

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

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

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

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

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

US Patent Nos.: Not Applicable.

Doc. CWQLHX520
April 11, 2005

Table Of Contents

QALoad online help	1
Getting started with QALoad	2
Welcome to QALoad	2
The load testing process	2
Developing scripts	3
Setting up the Conductor	3
Validating scripts	4
Running a load test	4
Analyzing test results	6
Script Development Workbench	10
Overview of the Script Development Workbench	10
About the Script Development Workbench	11
Sessions	12
Developing a test script	15
Using EasyScript	66
NetLoad	182
UNIX	186
Testing with QARun	187
Troubleshooting	188
Conductor	191
Overview of the QALoad Conductor	191
About the Conductor	191
Setting up a test	202
Running a test	213
Running a series of tests (batch)	216
Monitoring a running test	216
Recording and replaying a test	219
Analyzing load test data	220
Integration and server monitoring	221
Troubleshooting	231
Player	239
Overview of the QALoad Player	239
About the Player	239
Dialog box and field descriptions	240
How to	242

Analyze.....	245
Overview of QALoad Analyze.....	245
About Analyze	245
Accessing test data.....	249
Displaying detail data.....	251
Creating a chart or graph	255
Customizing a chart or graph.....	257
Viewing reports.....	258
Publishing or sharing test results	267
Language Reference	271
Contents of QALoad Language Reference.....	271
ADO	271
Citrix.....	401
DB2	423
ODBC	435
ODBC/DB2.....	438
Oracle 7.....	471
Oracle 7/8	491
Oracle 8.....	495
Oracle Forms Server	525
QALoad.....	619
QARun integration	645
SAP 4.x	646
SAP 6.x	657
SSL.....	671
Tuxedo	676
Uniface.....	698
Uniface Polyserver (Versions 7.2.04 - 7.2.06).....	722
Winsock	762
WWW	791
Compuware customer support	864
Contact information.....	864
World Wide Web Information	864
QALoad glossary	865
Non-alphabetic.....	865
A.....	865
B.....	865

C.....	865
D.....	865
E.....	865
F.....	865
G.....	866
H.....	866
I.....	866
J.....	866
K.....	866
L.....	866
M.....	866
N.....	866
O.....	866
P.....	866
Q.....	866
R.....	866
S.....	867
T.....	867
U.....	867
V.....	867
W.....	867
X.....	867
Y.....	867
Z.....	867
Accessibility features.....	870
Accessibility of QALoad components.....	870
Accessibility of QALoad documentation.....	870
Assistive technology tools that enhance the accessibility of QALoad.....	871
Product shortcut keys.....	871
Compuware customer support.....	874
Contact information.....	874
World Wide Web Information.....	874
Index.....	875

QALoad 5.02

[QALoad online help](#)

Getting started with QALoad

Welcome to QALoad

With QALoad, you can simulate the load generated by thousands of users without the expense of actual end users or their equipment. QALoad enables you to quickly develop test scripts, control the conditions for the test, create the virtual users that simulate the load, initiate and monitor the test, and report the results.

The following components of QALoad control different aspects of the load testing process:

Script Development Workbench

Develop your scripts. Capture sessions and convert the transactions into C, C++, or Java-based script.

Conductor

Control session activity. Initiate and monitor the test.

Player

Simulate user activity based on commands given from the Conductor.

Analyze

View summary report data and create other statistical reports from the test.

Before you begin using QALoad, review the [Getting started](#) section of the online help to familiarize yourself with load testing concepts.

The load testing process

You begin the testing process by determining the types of application transactions you want to emulate. You then develop these transactions into QALoad scripts by creating the same types of requests that your applications invoke on the server. Each transaction becomes its own script. The QALoad Script Development Workbench lets you easily create full-function scripts.

When you plan your test, you need to decide which transactions to run, the number of simulated clients that will run each transaction script, and the frequency at which each script will run. When you run the test on a workstation with the QALoad Player component, you can specify transaction rates as well as fixed and random delays to better emulate real-system activity. QALoad considers these factors a test scenario and stores them in a session ID file.

While a test is running, the test operator can dynamically view overall run times as well as individual transaction performance. QALoad's Conductor component collects this data for analysis at the conclusion of the test.

After executing the test, summary reports show the response times that the emulated clients experienced during the test. Individual and global checkpoints let you view and identify specific areas of system performance. You can export all test output data to spreadsheet and statistical packages for further analysis or use QALoad's Analyze component to create presentation-quality reports and graphs.

As shown in the following image, a typical load test setup consists of a QALoad Conductor, one or more QALoad Players, and the system under test.

QALoad Conductor

QALoad's Conductor controls all testing activity such as setting up the session description files, initiating and monitoring the test, and reporting and analyzing test results.

QALoad Player

A QALoad Player creates virtual users that simulate multiple clients sending middleware calls to a server under test. In a typical test setup, one or more QALoad Player workstations run under any Windows 32-bit platform (Windows XP or 2000) or UNIX. For large tests (thousands of simulated clients), you can connect multiple Players to QALoad's Conductor. The Conductor and Players communicate using TCP/IP.

The hardware and software capabilities of the Player machine are the only factors that limit the capacity of an individual QALoad Player. The maximum number of virtual users per Player machine is dependent on the system under test, the characteristics of the script, and the test scenario. You can specify how many threaded- and process-based virtual users to assign on a machine on the [Machines tab of the Options dialog box](#) in the Conductor. The Conductor calculates how many virtual users will be active per 64 MB of RAM, based on the values you provide in these fields.

System under test

The servers you test are typically production systems or a duplicate of a production system that is set up at a test facility. If you perform any kind of system selection or performance stress test, the system under test must use the same hardware and software (including current versions) as the production environment. Compuware has found that even subtle changes have profound effects on performance.

Developing scripts

The QALoad Script Development Workbench is used to develop test scripts. You use the QALoad Script Development Workbench to develop scripts. It contains facilities for capturing sessions, converting captured sessions into scripts, and modifying and compiling scripts. Once you compile your script, you can use QALoad's Conductor and Player components to test your system.

Record Facility: QALoad's Record facility, which you can access through the QALoad Script Development Workbench, records the transactions that your terminal, browser, or client makes. It stores these transactions in a capture file.

Convert Facility: QALoad's Convert facility, which you can access through the QALoad Script Development Workbench, converts capture files into scripts. It generates a one-to-one correspondence of transactions from the original session to your QALoad script.

Visual Navigator: Visual Navigator for WWW is QALoad's easy-to-use visual interface to QALoad's powerful script development tools. Visual Navigator for WWW renders your recorded C-based transaction in a tri-paned, browser-like environment similar to popular visually-oriented development tools, with icons representing all the elements of your script.

Using QARun scripts for load testing

QALoad provides you with the functionality to perform load tests using your QARun scripts. By inserting your QARun script into a QALoad script template, you can time your GUI-driven business transactions and include those timings in QALoad post-test reports.

Setting up the Conductor

To prepare for running a load test, you must set up the Conductor:

1. [Start the Conductor.](#)
2. [Configure the Conductor.](#) After starting the Conductor, you may need to verify that the Conductor's configuration parameters are set properly.
3. [Set up a session ID file.](#) For every test you run, you must create a session ID file containing information the Conductor needs to run the test, such as which scripts to run, which Player machines to use, and whether to collect

server or performance monitoring data. You use the Conductor to create and save session ID files in the `\QALoad\Session` directory.

Server and performance monitoring

QALoad integrates several mechanisms for merging load test response time data with server utilization data and performance metrics. Most methods produce data that is included in your load test timing results and processed in QALoad Analyze. The only exception is ApplicationVantage. Data captured from ApplicationVantage can be opened in ApplicationVantage or Application Expert, but not in QALoad.

If you plan to collect server and performance monitoring data, you must set the appropriate options in the Conductor before running the load test. The following methods of server and performance monitoring are available:

- ! [Remote Monitoring](#) — allows you to monitor server utilization statistics from a remote machine without installing any software on the remote machine.
- ! [Server Analysis Agents](#) — must be installed on each applicable machine.
- ! [ServerVantage](#) — integrates with your existing ServerVantage installation. You must be licensed for and have installed and configured the appropriate product in order to integrate with QALoad .
- ! [Application Expert/ApplicationVantage](#) — collects test data that you can open in both Application Expert and Application Vantage.

Validating scripts

Before you conduct an actual load test, you should individually validate the script(s) you plan to use in the load test by running it in a simple test. If the script runs to the end without any errors and runs multiple times without errors, it is valid to use in a load test.

If the script aborts on an error, debug the script and run it through a simple test again. You can validate scripts from the QALoad Script Development Workbench, the QALoad Player, or the QALoad Conductor.

Running a load test

Running a load test

After validating a script using one of the methods described in [Validating scripts](#), it is safe to run a load test with that script. See the following topics for more information:

- ! [Preparing for a Load Test](#)
- ! [Starting a Load Test](#)
- ! [Monitoring a Load Test](#)
- ! [Stopping a Load Test](#)

Preparing for a load test

Before you run a load test, you must complete the following tasks:

- ! **Prepare a datapool file:** If you created a datapool file using the QALoad Script Development Workbench, QALoad stores the file where the Conductor can automatically access it. However, if you created a datapool file using a text editor (for example, Notepad), you must place the file in your appropriate `\Middlewares\<<middleware_name>\scripts` directory (for example, `\QALoad\Middlewares\Oracle\Scripts`) so the Conductor can access the file.

For information about datapool files, see [Simulating user-entered data](#).

- ! **Set Up SSL Client Authentication for Virtual Users (SSL scripts only):** If you are running a load test with a WWW script containing SSL requests, you should export a Client Certificate from your browser into QALoad or create a QALoad 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 export Client Certificates from your browser and convert them for use in QALoad or to create QALoad Client Certificates, see [Importing a client certificate from a Web browser](#).

Once you export or create the necessary Client Certificates, you can insert them into your script using datapools.

Starting a load test

While a load test is running, the Conductor's toolbar changes from the Configuration and Setup Toolbar to the [Runtime Toolbar](#). The Runtime Toolbar buttons let you control the test and access detailed information about the test while it is running.

For more information about what to expect from the QALoad Conductor while a test is running — including descriptions of the Runtime Toolbar buttons — see [Monitoring a load test](#).

To start a load test, click the Run button on the configuration and setup toolbar or choose Start Test from the Conductor's Run menu.



Note: While any window on the desktop is re-sizing or re-positioning, all Windows applications pause. Do not click and hold on a window caption or border for extended periods during a load test because it delays message handling and may impact the test results.

Monitoring a load test

When a test is started, the QALoad Conductor's interface changes to an interactive test control station, referred to as the [Runtime Window](#). The Runtime Window displays information about the scripts, machines, and virtual users that are executing the load test. From the Runtime Window, you can observe the progress of individual scripts and Player machines, create and view real-time graphs, and start or suspend scripts and Players from a running test to better simulate the unpredictability of real users.

In addition to the test data shown by default on the Runtime Window, you can access detailed test information using the QALoad Conductor's [Runtime toolbar buttons](#). You can:

- ! View statistics for a single virtual user
- ! View the activities of a virtual user in a browser-like window (WWW only)
- ! Step to the next request (WWW only)
- ! View the current datapool record
- ! Display the script running on a single virtual user
- ! Display messages sent from a Player workstation to the QALoad Conductor
- ! Display statistics about Conductor/Player communication
- ! Show/hide the Runtime Tree or Runtime Control Panel
- ! Synchronize all virtual users
- ! Exit, abort, or quit the test

Running a batch test

By setting the appropriate options in the Conductor, you can elect to run a series of tests as a batch, rather than one at a time. A batch test is comprised of multiple session ID files that are executed sequentially.

You can create a batch test by adding a number of session ID files to a batch file. Before you can add a session ID to a batch file, the following conditions must be true:

- ! The session must include a defined number of transactions. Sessions of unlimited transactions cannot be used in a batch test.
- ! All scripts to be included must exist prior to starting the batch test. This means the .c files referenced in the selected session ID files must be present in the scripts directory.

Stopping a load test

A load test is complete when all virtual users exit. A virtual user automatically exits when one of the following occurs:

- ! A script encounters an EXIT command.
- ! A script completes its transaction loop.

To stop a load test, click the Exit button.

Adding post-test comments

If you selected the Display Post Test Comments option on the General tab of the Options dialog box when you configured the Conductor, the Post Test Comments window opens when you click the Quit button. Type any comments, which are saved to the test's Summary Report, which can be viewed in QALoad Analyze.

Analyzing test results

Analyzing test results

After you set up a load test and run it, you can analyze the results from the test using QALoad Analyze.

An important part of the load testing process is viewing and studying the results of a test. You can view load test results not only on a machine where QALoad is installed, but also on any machine with a Web browser.

When you run a test using a particular session ID file (set up in the Conductor), each Player compiles a local timing file comprised of a series of timing records for each checkpoint of each script run on that Player. Each timing record in the file consists of a response time/elapsed time pair of values specifying the amount of time it took a certain checkpoint to finish (response time) at a specific time in the test (elapsed time).

At the end of a test, Player timing files are sent to the Conductor and are merged into a single timing file, called the Primary timing file, for analysis. If you set up integration with Compuware's ServerVantage product, the Conductor collects timing data from the ServerVantage central console and also merges that data into the timing file.


When you open a timing file, QALoad generates a working folder that contains all supporting files, reports, and images generated from that timing file. The folder is located in the `\Program Files\Compuware\QALoad\TimingFiles\xxx.xml` source directory, where `<xxx>` is the name of the timing file.

Custom reports

QALoad Analyze provides the ability to create custom reports using XML (Extensible Markup Language), XSL (Extensible Style Language), and HTM (Hypertext Markup) files. QALoad Analyze provides a set of files in .htm, .xml, and .xsl formats in addition to the .tim file. QALoad Analyze automatically generates a XML (*.xml), XSL (*.xsl), and HTM (*.htm) file when you open a timing file.

Pre-defined reports

QALoad Analyze provides pre-defined reports so you can receive immediate load test results without having to manipulate any data. All the files necessary for those reports are located in the directory `\Program Files\Compuware\QALoad\Timing Files\Reports`.

 **Note:** The pre-defined reports that are available depend on the data collected in the timing file, which is determined by the QALoad Conductor option you select at the time of running the load test.

Graphing data

Starting with the Workspace, you can use QALoad Analyze's charting features to graph timing data in a number of formats and styles.

Managing large amounts of data

With a large number of virtual users, it's possible to create a timing file containing hundreds of thousands of timing records for each checkpoint. Attempting to graph just a few of those checkpoints can slow QALoad Analyze down considerably. For example, if a timing file contained 250,000 timing records for each data point, attempting to graph even one checkpoint means that QALoad Analyze has to paint 250,000 lines on the graph.

Since most monitors only have 1024 pixels across the screen, the 250,000 data points would mostly be plotted atop one another and the results would be unreadable.

Now imagine attempting to graph the data of several data points of that size. The sheer amount of data could easily overwhelm a workstation. And every time you move the window, resize the window, right-click on the graph, or so on, QALoad Analyze has to re-draw the graph. You could conceivably spend enormous amounts of time simply attempting to graph data.

To make large amounts of data manageable, QALoad Analyze provides an option that allows you to determine how to thin data. That is, how to determine how many data points to plot.

QALoad Analyze graph types

The following basic graph types are available from QALoad Analyze. After generating one of the following graph types, you may further customize a graph in a number of ways.

Line Graph

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

Bar Graph

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

Transaction Throughput Graph

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

Response Time Distribution Graph

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

Cumulative Response Time Distribution Graph

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

Customizing graphs

You can also change the style and appearance of a graph using options available from the Graph toolbar. Display the Graph toolbar by right-clicking in an open area of a graph and choosing the Toolbar option from the shortcut menu. The Graph toolbar contains buttons for standard Windows operations as well as for customizing the appearance of your graphs.

The Graph toolbar includes the following features for customization:

- ! Graph type (gallery type)
- ! Color
- ! Grid orientation (horizontal and vertical)
- ! Legend box
- ! Data display
- ! Dimension (3D or 2D)
- ! Rotation
- ! Z-Cluster
- ! Color/pattern

Integrating ServerVantage agent data


If you set options to integrate ServerVantage resource utilization data before running a test, that data is included in the resulting timing file and can be sorted and displayed in QALoad Analyze in much the same way as QALoad timing data. ServerVantage data provides a summary of all the Agents that ServerVantage monitored during the load test and details aggregate statistics for Agent data points including minimum, maximum, and

mean data values.

Displaying ServerVantage Agent data

When you open a timing file containing ServerVantage Agent data, QALoad Analyze displays test data with QALoad timing data two ways:

- ! ServerVantage Agent workstations are listed in the Server Monitoring group in the Workspace tree-view, under the Resource Trends (ServerVantage) branch. From the Workspace, select **Agent workstations** to create detail or graphical views of the Agent data points. Specifically, you can:
 - Display Agent data point details.
 - Graph Agent data point details.
- ! Detailed data point information is displayed in the Data window. The ServerVantage detail view includes data such as the name of the machine where you ran the ServerVantage Agent; the Agent name; and the minimum, maximum, and mean data values for the Agent.

 **Note:** ServerVantage resource utilization data is available only if you set the ServerVantage integration options on the QALoad Conductor's Test Information window before executing a load test.

Viewing Application Expert and QALoad integrated reports

QALoad integrates with Application Expert version 8.0 and 9.0 to help analyze network performance during a load test. Application Expert is a Windows-based tool that enables users to examine the effects the network will have on the performance of new or modified applications prior to live deployment. Application Expert provides reports that help network managers identify poorly performing applications.

When using the Application Expert integration in a load test, QALoad generates a trace file. This file is the capture file created by the ApplicationVantage Agent. The name of the trace file is <Session>_<YYYYMMDD>_<HHMMSS>.opt/.opx where <Session> is the name of the QALoad Conductor session used to execute the load test, and <YYYYMMDD> and <HHMMSS> are the date and time the trace file was captured. The trace file extension for Application Expert version 8.0 is .opt; for version 9.0 it is .opx.

It is located in the (default) directory \Program Files\Compuware\QALoad\LogFiles. At the end of a load test, a high-level static report and various supporting files are automatically generated from the trace file and located in the directory \Program Files\Compuware\QALoad\LogFiles\<Session>_<YYYYMMDD>_<HHMMSS>.

You can view the static report at any time in a Web browser such as Microsoft Internet Explorer. The static report contains the following Vantage views:

- ! Performance Overview
- ! Network Utilization and Transit Time
- ! Node Processing Detail
- ! Node Sending Detail
- ! Bounce Diagram
- ! Error Analysis
- ! Thread Analysis
- ! Conversion Map

A description of each view is provided in the <Session>_<YYYYMMDD>_<HHMMSS>.xml file.

To generate the trace file, ApplicationVantage Agent must be installed on the same machine as QALoad Conductor. The Agent can be installed during the QALoad install or installed independently.

To generate the high-level static report, Application Expert or Application Vantage must be installed on the same machine as QALoad Conductor. For additional test analysis, import the trace file into Application Expert or ApplicationVantage. To use Application Expert or Application Vantage to further analyze the trace, refer to the Application Expert or ApplicationVantage User's Guides or online help.

Script Development Workbench

Overview of the Script Development Workbench

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

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

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

In short, to produce a usable test script you will:

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

To access the Script Development Workbench, click Start>Programs\ Compuware\ QALoad\ Script Development Workbench. To begin developing a test script, select the appropriate middleware from the Session menu.

Specify your middleware type by choosing the middleware name from the Session menu or clicking the appropriate button on the toolbar. The Default Session Prompt opens. If this middleware type should be the default every time you open the Script Development Workbench, select the check box Make this my default session.

If you do not want to be prompted to set a default middleware, clear the Enable default session checking check box. You can also turn default session checking on or off from the [Configure Script Development Workbench dialog box](#) at any time.

Click OK.

To set up automatic conversion and compilation:

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

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

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

About the Script Development Workbench

Overview of the Script Development Workbench

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

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

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

In short, to produce a usable test script you will:

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

To access the Script Development Workbench, click Start>Programs\ Compuware\ QALoad\ Script Development Workbench. To begin developing a test script, select the appropriate middleware from the Session menu.

Specify your middleware type by choosing the middleware name from the Session menu or clicking the appropriate button on the toolbar. The Default Session Prompt opens. If this middleware type should be the default every time you open the Script Development Workbench, select the check box Make this my default session.

If you do not want to be prompted to set a default middleware, clear the Enable default session checking check box. You can also turn default session checking on or off from the [Configure Script Development Workbench dialog box](#) at any time.

Click OK.

To set up automatic conversion and compilation:

1. From the **Script Development Workbench** menu, choose **Options>Workbench**.
2. On the Configure Script Development Workbench dialog box, in the **Record Options** area, select the check box **Automatically Convert Capture**.

3. Click the **Compiler Settings** tab.
4. Select the check box **Automatically compile scripts**.
5. To ensure that a script isn't overwritten accidentally, select the check box **Prompt before overwriting script**.
6. Click **OK** to save your settings.

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

Setting a default middleware session

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


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

Configuring the Script Development Workbench

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

To set a working directory:

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

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

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

Sessions

EasyScript Sessions

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

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

You can also open a [Universal session](#) to record calls from multiple middlewares within a single session.

Using the Universal session


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

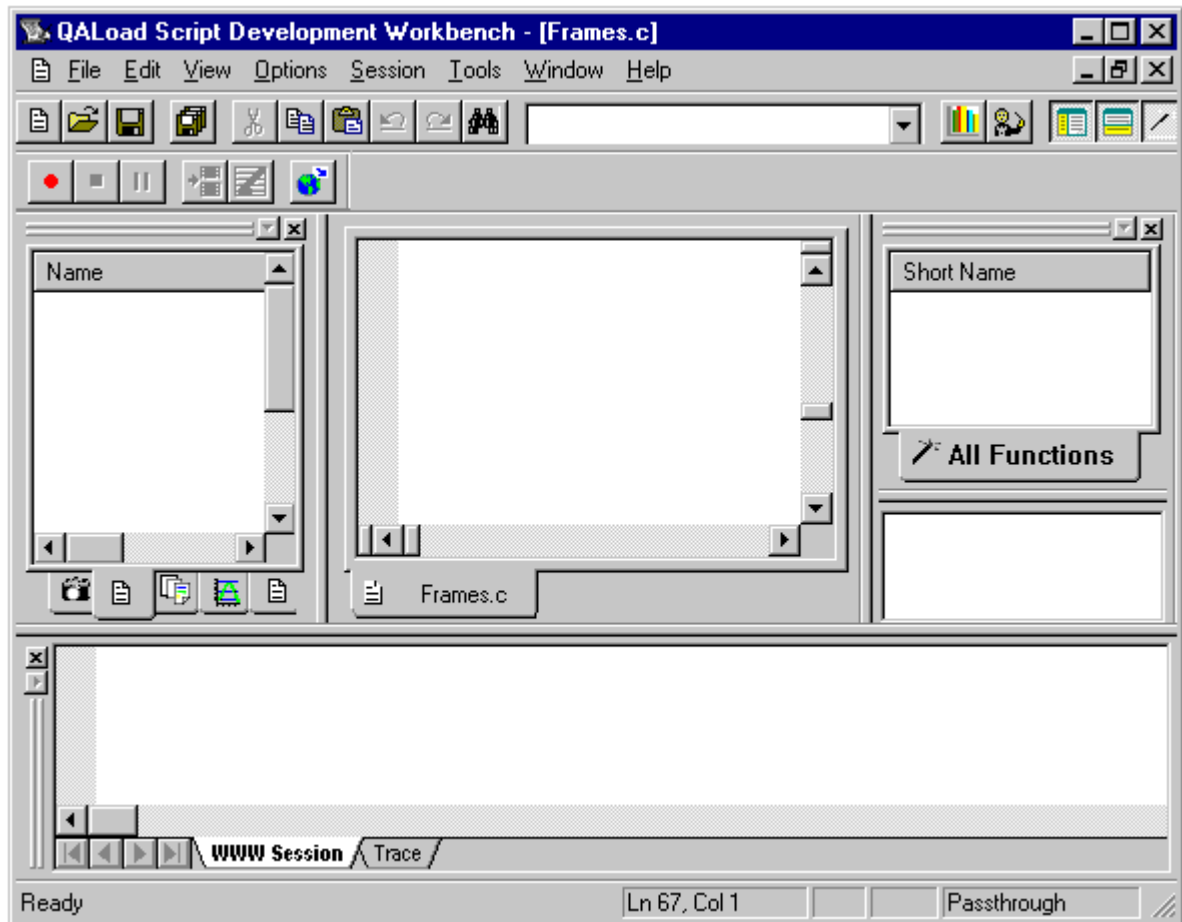
For example, your browser might download and open a Java applet which then communicates with a Winsock server. If you recorded that activity using a simple WWW session, the Script Development Workbench would only record the HTTP requests that downloaded and opened the Java applet. Recording that transaction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

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

The Script Development Workbench main window

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

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



Menus and toolbar buttons

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

Menus and toolbars without an open EasyScript session

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

File
View
Options
Session
Tools
Help

Toolbar Buttons

Menus and toolbars with an open EasyScript session

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

File
Edit
View
Options
Session
Tools

Window
Help

Toolbar Buttons
Recording toolbar

Recording toolbar

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

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



Developing a test script

Recording a transaction

Recording middleware calls

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

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

To record a middleware call:

1. Open an appropriate middleware session in the QALoad Script Development Workbench.
2. (Oracle Forms Server only) Choose **Options>Workbench**, then click **Java** to set the location of your Java files for recording.
3. Select **Session>Record>Start**. The Record Options wizard opens.
4. If necessary, set or change any recording options on your middleware-specific options tab. Press **F1** from the middleware options tab for a description of available options.
5. From the toolbar, click **Start Record**. QALoad launches your application and any proxies, if necessary, and begins recording any calls.
6. Run the desired user operations using your application.
7. (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.
8. (Optional) At any time during the recording process, you can [insert any necessary commands or comments into the capture file](#).
9. When you have recorded a complete transaction, stop the application from which you are recording.

10. When you finish, click **Stop Record**. You will be prompted to save your capture file. By default, capture files (.cap) are saved in the QALoad\Middlewares\

 **Note:** If QALoad is not able to record from your application, try QALoad's [alternate procedure](#) for recording.

Inserting commands/comments into a capture file

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

To insert commands/ comments into a capture file:

1. On the Recording toolbar, click **Insert Command**. The toolbar expands into a window where you can select options for inserting commands into your capture file.
2. In the **Command Type** area, select whether you want to insert a comment or a begin/end checkpoint.
3. In the **Command Info** area, type your comment or a description of the checkpoint.
4. Click **Insert** to insert your comment or checkpoint command into your capture file, and then continue recording your transaction as usual.


Converting a transaction to a script

Converting

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

To convert a capture file to a script:

1. Access the QALoad Script Development Workbench. [Details](#).
2. From the **Session** menu, choose the session you want to start.
3. If you have not already done so, [set conversion options](#).
4. In the **Workspace Pane**, click the **Captures** tab.
5. Click on the capture file you want to convert and click **File>Convert**.
6. If an Error/Warning Summary opens in the **Output Pane**, resolve any errors if necessary.
7. Compile the script.

 **Note:** You can set an option to automatically convert your recorded transactions into scripts. [How?](#)

Compiling a test script

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

To learn more about compiling a script, click one of the following options:

[Setting up automatic compiling](#)

[Setting advanced compiler options](#)

[Compiling Win32 Scripts](#)

[Compiling UNIX Scripts](#)

Customizing a script

Using the Function Wizard


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

The Function Wizard is located in the Script Development Workbench in a pane on the right side of the window that you can enable or disable from the View menu.

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

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

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

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

Using custom counters and messages

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

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

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

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

The following sample script illustrates both script counters and messages:

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

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

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

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

QALoad 5.02

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

SYNCHRONIZE();
BEGIN_TRANSACTION();

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

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

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

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

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

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

Defining checkpoints

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

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

Defining transaction loops

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

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

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

Simulating user-entered data

When you create a script, you 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 QALoad Script Development Workbench to create, maintain, and use datapool files (.dat) to insert variable data into your scripts.

A datapool can be defined as either central or local:

- ! A central datapool is a datapool that resides on the same workstation as the QALoad Conductor, and is available to any Player system on the network that requests it from the QALoad Conductor. A central datapool is controlled by the QALoad Conductor, and you use the QALoad Conductor to set any options relating to a central datapool.
- ! 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 QALoad Script Development Workbench to insert local datapools into a script.

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

Creating a datapool file

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

To create a datapool file:

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

Modifying a datapool file

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

To modify a datapool file:

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

Using a central datapool file in a script

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

Assigning a central datapool file

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

Using data records from a central datapool file

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

To add datapool statements to your script:

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

Using local datapool files in a script

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

Assigning a Local Datapool

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

`CLOSE_DATA_POOL` at the default locations in the script. These statements will be bookmarked in the margin for easy identification.

12. When you are finished modifying the script, save any changes.

For detailed information about any of these commands, refer to the [Language Reference](#) section.

Using Data Records from a Local Datapool File

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

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

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

For detailed information about any of these commands, refer to the [Language Reference](#) section.

Inserting Variable Data with ActiveData Substitution

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

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

1. Start the appropriate session in the QALoad Script Development Workbench.
2. In the Workspace pane, click the **Scripts** tab.
3. On the **Script** tab, double-click the script you wish to open. The script opens in the Workbook pane.
4. In the script, highlight the string you wish to replace.
5. Right-click anywhere in the highlighted string.
 - ! To substitute a value from a datapool:
 - — Click **ActiveData>Datapool Substitution** in the shortcut menu that opens. The **ActiveData Datapool Substitution** dialog box opens.
 - In the **Datapool(s)** area, highlight the datapool to use. The contents of the datapool file display below. If the datapool you want to use is not listed, click the **Add** button to add it to the list of available datapools.
 - In the **Field: ID** field, type the field number of the specific value to use from the datapool.
 - When you are finished, click **OK**. The QALoad Script Development Workbench will place a `#define` statement identifying the datapool file at the beginning of your script. It will also insert the datapool commands `OPEN_DATA_POOL`, `READ_DATA_RECORD`, `GET_DATA_FIELD` and `CLOSE_DATA_POOL` at the default locations in the script, and

bookmark them in the margin for easy identification. Refer to the [Language Reference](#) section for detailed information about any of those commands.

- To substitute a variable:
 - Click `ActiveData>Variable Substitution` from the shortcut menu that appears. The `ActiveData Variable Substitution` dialog box opens.
 - Assign a variable name for the selected string in the `Variable Name` field.
 - Click `OK`. The QALoad Script Development Workbench will declare the variable at the beginning of your script and substitute the named variable for the selected string. It will also bookmark both locations for easy identification.
6. When you are finished, save your script changes. Compuware recommends that you also compile your script to check for any errors.

Middleware scripting techniques

Citrix

Handling dynamic windows

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

Using the `CtxWaitForScreenUpdate` command

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

Handling dynamic window titles

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

1. **Ensure that the `Enable Wildcard Title Match` check box is selected in the Citrix conversion options prior to converting the recording.**
In the `Window Verification` group of the **Citrix Convert Options** dialog box, ensure that the **Enable Wildcard Title Match** check box is selected. This check box is selected by default. If you are working with a previously-converted script, ensure that a `CtxSetEnableWildcardMatching` command exists in the script prior to the `BEGIN_TRANSACTION` command and that the parameter is set to `TRUE`.
2. **Verify whether there is an issue with dynamic window titles.**
When a script fails on validation because the run time window title is different than the expected window title from the recording, it is likely that you are dealing with a dynamic title issue that can be handled by this scripting technique. In this case, the script fails on the `CtxWaitForWindowCreate` call.
3. **Identify a match “pattern” for the dynamic window title.**
Note the error message that is returned during validation (or replay). The message indicates the expected window title versus the window title from script playback. Examine the differences in the window titles to create a “match pattern” that recognizes the window title, while ignoring other windows. A match pattern can be a simple substring of the window title or a pattern string using wildcard characters such as `?` (to match any single character) or `*` (to match any number of characters). The examples below illustrate the different match patterns.

4. **Insert a CtxSetWindowMatchTitle command prior to the CtxWaitForWindowCreate call for the dynamic window.**
When adding the SetWindowMatchTitle command, ensure that the first parameter contains the correct window object and the second parameter contains the match string in double-quotes.
5. **Validate the script to ensure the CtxWaitForWindowCreate command recognizes the dynamic window name.**
Run the revised script through validation to ensure that the script succeeds. If the script does not validate successfully, go to step 3 to determine if the match pattern is correct.

Example 1: Using a substring match

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

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

Example 2: Using a wildcard match with the * character

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

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

Example 3: Using a wildcard match with the ? character

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

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

Handling dynamic windows that require user interaction

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

To work around this issue, follow these steps:

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

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

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

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

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

Moving the Citrix connect and disconnect outside the transaction loop

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

Recording

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

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

Scripting

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

Also comment out the DO_SetTransactionStart and DO_SetTransactionCleanup calls.

Handling Citrix server farms

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

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

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

.
.
.
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("Orders.cpp");

CitrixInit(4);

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

SYNCHRONIZE();
```

Handling unexpected events in Citrix

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

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

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

Example

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

```
/*
 * capSave11111-2.cpp
```

QALoad 5.02

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

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

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

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

#include "do_citrix.h"

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

#ifndef NULL
#define NULL 0
#endif

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

    /* Citrix Window Information Objects */
    CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);
    CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
    CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
    CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198,
397, 127);
    CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198,
397, 127);
    CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
    CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
    CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
    CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
}
```



```

CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
CtxWI *CWI_11 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
CtxWI *CWI_12 = new CtxWI(0x2006a, "", 200, 186, 156, 287);
CtxWI *CWI_13 = new CtxWI(0x10138, "", 112, 116, 416, 248);
CtxWI *CWI_14 = new CtxWI(0x50036, "Microsoft Word", -4, -4, 649, 461);
CtxWI *CWI_15 = new CtxWI(0x1017e, "Open", 19, 23, 602, 387);
CtxWI *CWI_16 = new CtxWI(0x20174, "*Microsoft Word", -4, -4, 649, 461);
CtxWI *CWI_17 = new CtxWI(0x10058, "", 113, 114, 305, 26);
CtxWI *CWI_18 = new CtxWI(0x2013e, "Calculator", 66, 66, 261, 253);
CtxWI *CWI_19 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
CtxWI *CWI_20 = new CtxWI(0x3006a, "Shut Down Windows", 111, 96, 418, 193);

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

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("capSave11111-2.cpp");

CitrixInit(1);

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

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

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

CtxWaitForWindowCreate(CWI_1, 2125);

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

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

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

.
.
.

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

CtxWaitForWindowCreate(CWI_14, 141);

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

```

QALoad 5.02

```
// Window CWI_13 ("") destroyed 1087837397.468

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

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

// Window CWI_15 ("Open") created 1087837400.764

CtxWaitForWindowCreate(CWI_15, 344);

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

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

DO_MSLEEP(109);

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

// Window CWI_15 ("Open") destroyed 1087837404.827

if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE,3000,CWI_16))
BeginBlock();
    CtxPoint(337, 265); //1087837404.905
    // Window CWI_16 ("11111111 - Microsoft Word") created
1087837404.905
    CtxWaitForWindowCreate(CWI_16, 31);
    // Window CWI_14 ("Document1 - Microsoft Word") destroyed
1087837404.905
    DO_MSLEEP(7547);
    CtxPoint(628, 9); //1087837414.592
    DO_MSLEEP(2141);
    CtxClick(CWI_16, 281, L_BUTTON, NONE); //1087837414.873
    DO_MSLEEP(234);
    // Window CWI_16 ("11111111 - Microsoft Word") destroyed
1087837415.108
    CtxPoint(113, 93); //1087837418.779
    // Window CWI_17 ("") created 1087837418.779
EndBlock()

///ReadOnly Code Start

else
BeginBlock();
    // Window CWI_117 ("File In Use") created 1087840076.599
    CtxWaitForWindowCreate(CWI_117, 578);
    DO_MSLEEP(2360);
```

```

CtxPoint(358, 283); //1087840079.068

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

DO_MSLEEP(109);
// Window CWI_117 ("File In Use") destroyed 1087840079.458

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

CtxWaitForWindowCreate(CWI_118, 63);

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

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

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

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

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

    EndBlock();

///ReadOnly Code End

DO_MSLEEP(1063);

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

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

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

// Window CWI_7 ("") destroyed 1087837440.138

DO_SetTransactionCleanup();

CtxDisconnect();

END_TRANSACTION();

delete CWI_1; // "Warning !!"
delete CWI_2; // "Log On to Windows"
delete CWI_3; // "Please wait..."
delete CWI_4; // "Citrix License Warning Notice"
delete CWI_5; // "Citrix License Warning Notice"
delete CWI_6; // "UsrLogon.Cmd"
delete CWI_7; // ""

```

QALoad 5.02

```
delete CWI_8; // "ICA Seamless Host Agent"
delete CWI_9; // "Program Manager"
delete CWI_10; // ""
delete CWI_11; // ""
delete CWI_12; // ""
delete CWI_13; // ""
delete CWI_14; // "Microsoft Word"
delete CWI_15; // "Open"
delete CWI_16; // "11111111 - Microsoft Word"
delete CWI_17; // ""
delete CWI_18; // "Calculator"
delete CWI_19; // ""
delete CWI_20; // "Shut Down Windows"

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

CitrixUninit();

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

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

    CitrixUninit();

    EXIT();
}
```

Oracle Forms Server

Understanding the C++ script

Understanding the C++ script

Oracle Forms Server scripts are produced for Oracle Forms 4.5, 6.0, 6i, and 9i (Release 2 and later) recordings. The C++ script executes OFS-related statements by passing the statements in the script DLL to the OFS Java engine that performs the client activities and the client communication with the server. Because the C++ script statements are directly tied to corresponding methods in the OFS Java engine, modifications to the script statements are limited to changing the property parameter values through variablization.

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

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

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

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

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

The following topics describe the three main sections of an Oracle Forms Server script in more detail:

[Connection statements](#)
[Application statements](#)
[Disconnect statements](#)

Connection statements

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

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

Server-side recording connections

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

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

HTTP connections

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

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

For example:

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

HTTPS connections

HTTPS connection mode uses the same connection statements as HTTP mode. To connect, QALoad internally performs the same tasks as the HTTP connection mode plus it performs the SSL connection

when the `ofsHTTPDoSSLHandshake` function is called. This statement is positioned in the script before the `ofsHTTPConnectToFormsServlet` function.

Socket connections

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

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

Application statements

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

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

Parameters of a property statement:

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

1. **Captured control name.** If the name is not available, this value is the class name to which the control belongs.
2. **Captured control ID.**
3. **Action type.** This flag indicates if the property is to be added to the current Forms Message or if the property ends the current Forms Message. During playback, each control is treated as a Forms Message. When the current Message ends, QALoad translates the control and its properties to binary format. The valid values are:
 - `OFS_ADD` – add the property to the current Message.
 - `OFS_ENDMSG` – add the property to the current Message and end the Message.
 - `OFS_STARTSUBMSG` – add the property of the succeeding nested Message to the current Message.
4. **Property ID.** The Forms version-specific ID of the property.
5. **Property value.** Captured value of the property (optional)
6. **Property value.** Captured value of the property (optional)

For example:

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

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

Forms environment statements:

The initial set of statements in the Forms script describes the Forms application environment. In this set, the "version" and the "cmdline" properties are the most important. The version property shows the Forms Builder version used by the application. The version indicates the capabilities of the application. For example, some versions cannot support HTTP connections. The cmdline property shows the Forms

configuration parameters passed to the server by the Forms applet. The parameter "record=names" indicates that the application enables GUI control names to be captured. Control names are preferred in multi-threaded playback. The "ICX" parameter indicates that the application uses a Personal Home Page, which requires that you supply OracleAppsLogin information on the [Oracle Forms Server Convert options dialog box](#) for the script to run successfully.

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

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

Sending messages to the server:

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

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

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

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

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

Getting the server reply:

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

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

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

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

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

Processing large data and delayed response scenarios:

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

Disconnect statements

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

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

Using script logging as a debugging tool

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

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


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

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

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

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

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

See Also

[Playback error codes for OFS](#)

Moving the OFS transaction loop

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

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

When modifying the script, use the comment lines as guides in moving the END_TRANSACTION() and BEGIN_TRANSACTION() statements. Ensure that there is a contextual flow from the new position of the

END_TRANSACTION() statement to the new position of the BEGIN_TRANSACTION() statement. The set of GUI objects that belong to the ofsSendRecv() statement just before the new END_TRANSACTION() statement must be the same as the set of GUI objects that belong to the ofsSendRecv() statement prior to the new BEGIN_TRANSACTION() statement.

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

The following example shows a modified OFS transaction loop:

New position of the BEGIN_TRANSACTION statement

```

/*
NewSales
*/

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

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

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

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

```

New position of the END_TRANSACTION statement

```

/*
EndTrans
*/

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

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

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

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

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

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

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

```

 Tips:


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

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

OFS and WWW Universal sessions

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

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

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

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

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

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

To use [ofsSetICXTicket](#), you must modify the script.

To capture an Oracle Applications login with [ofsSetICXTicket](#):

1. Add the following variable declaration statements to the top of the script:


```
char *p;
char ICX_Ticket[100];
char *pTicket;
```
2. In the `*.postcapweb` file, find the HTTP request that returns the ICX ticket. The reply should contain a string that indicates the ICX ticket value, such as "ICX_TICKET=". Note the left and right characters that delimit the ICX ticket value. In the example in step 4, the left delimiter is "icx_ticket='" and the right delimiter is "'".
3. In the script, find the matching request line for the HTTP request.
4. After the matching HTTP request line, add the `DO_GetUniqueString` statement using your chosen delimiters. For example:


```
p = DO_GetUniqueString( "icx_ticket='", "'");
```
5. Add script lines that copy the extracted value into your script variables.


```
strcpy(ICX_Ticket, p);
pTicket=ICX_Ticket;
```

QALoad 5.02

- (optional) Verify the ICX ticket value.
`RR__printf("ICX_Ticket=%s\n", ICX_Ticket);`
- Add the script line that passes the value of the ICX ticket to the OFS statement of `ofsInitSessionCmdLine`.
`ofsSetICXTicket(&pTicket);`

SAP 6.x

Required commands

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


```
SET_ABORT_FUNCTION(abort function);
DEFINE_TRANS_TYPE("capture.cpp");
HRESULT hr = CoInitialize(0);
if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info, "ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
```

Required commands for transaction restarting

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

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

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

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

Error handling and reporting

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

```
BEGIN_TRANSACTION();
try{
    SAPGuiConnect( s_info, "qacsapdb2");
    SAPGuiVerCheckStr("6204.119.32");

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

    DO_SLEEP(3);

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

    DO_SLEEP(6);

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

```

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

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

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

...

DO_SLEEP(10);

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

} // end try

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

END_TRANSACTION();

```

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

```

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

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

SAPGuiSetCheckScreenWildcard('');

SYNCHRONIZE();

BEGIN_TRANSACTION();

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

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

END_TRANSACTION();

```

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

```

SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("capture.cpp");
HRESULT hr = CoInitialize(0);
if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info,"ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
SAPGuiConnect( s_info,"qacsapdb2");
SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
SAPGuiCmdl(GuiTextField,PutText,"qaload1");
SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
SAPGuiCmdlPwd(GuiPasswordField,PutText,"~encr~1211616261");
SAPGuiCmd0(GuiPasswordField,SetFocus);
SAPGuiCmdl(GuiPasswordField,PutCaretPosition,3);
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmdl(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", "SAPLSP01", "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(%)\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 logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this issue, you can use the `SAPGuiPropIdStrExists` and `SAPGuiPropIdStrExistsEnd` commands to handle either scenario. This technique works by checking for the multiple logon dialog box from SAP and selecting the Continue option.

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

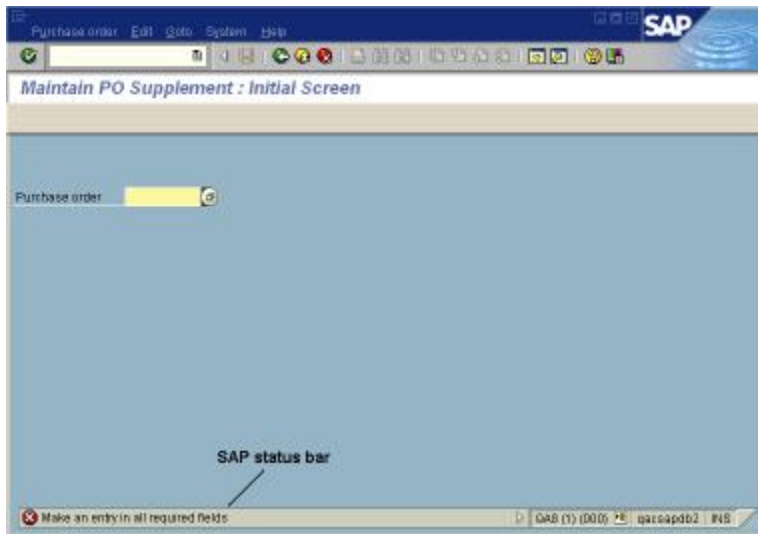
```

...
SAPGuiCheckScreen("S000", "SAPMSYST", "SAP");
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");
    DO_SLEEP(24);
    SAPGuiCmd0(GuiRadioButton,Select);
    SAPGuiCmd0(GuiRadioButton,SetFocus);
    SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");
    SAPGuiCmd0(GuiButton,Press);
    SAPGuiCheckScreen("S000", "SAPMSYST", "License Information for Multiple Logon");
SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");
...

```

Checking the SAP status bar

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



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

The `SAPGuiCheckStatusbar` command is used in the following script example:

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

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

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

if (bRetSts)
    RR__printf(" True\n");
else
    RR__printf(" False\n");
...

```

Object life span

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

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

Tuxedo

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. QALoad 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 = tmalloc( "FML", "", 4096 );

```

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

```

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

```

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

To simplify buffer management and provide more comprehensive error checking, QALoad Tuxedo scripts automatically handle buffer management. Instead of having to work with buffer pointers, QALoad'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 Do_Tuxtpcall command.

```

Do_Tuxtppalloc( Buf1 , "FML", 1024 );
Do_TuxFinit( Buf1 );
Do_TuxFMLData( test_carray, 1, "abcdefg" );
Do_TuxFMLData( test_long, 1, "12345" );
Do_Tuxtppcall( "OPEN_TEST1", Buf1 , Buf2 , 0 );

```

In the example above, the Do_Tuxtppalloc 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_Tuxtppcall makes a service call to OPEN_TEST1. The parameter list of the Do_Tuxtppcall 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. QALoad 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_Tuxtppalloc command, QALoad checks to see whether the buffer associated with a specified buffer index was previously allocated. If QALoad 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 QALoad commands to pass output data from one Do_Tuxtppcall as input to another Do_Tuxtppcall.

```

/* 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_Tuxtppalloc( Buf1 , "FML", 1024 );

```

QALoad 5.02

```
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 retrieved 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`.

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 QALoad'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;
```

 **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.

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 QALoad's Convert facility places in the script or for data you place in the script.

The following QALoad 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~

Winsock

Understanding data representation in the script

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

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

1. The character is compared to the "space" character in the ASCII table, which has a decimal value of 32. If the character's value is less than 32, the following steps are taken:
 - b. If the character is "\r", "\n", "\t", or "\f", it is represented in the script as a normal C escape character.
 - c. If the character is either "\\" or "\^", it is represented in the script as an octal character. For example, the values would be "\034" and "\036", respectively.
 - d. If the character's value is less than 32 and it does not meet the descriptions in a) and b) above, it is represented in the script as a control character. For example, if the character is a null character, it is represented in the script as "^@".
2. If the character's decimal value is between 32 (the "space" character) and 126 (~), it displays in the script as a standard readable ASCII character, with the following exceptions:
 - If the character is "\", which has a decimal value of 92, it is represented as "\\\" in the script.
 - If the character is "\"", which has a decimal value of 34, it is represented as "\"\" in the script.

- If the character is “^”, which has a decimal value of 94, it is represented as “^^” in the script.
3. If the character has a decimal value of 127, which corresponds to Delete (DEL), it is represented as “^” in the script.

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

Code	Octal	Decimal	Char
^@	000	0	NUL
^A	001	1	SOH
^B	002	2	STX
^C	003	3	ETX
^D	004	4	EOT
^E	005	5	ENQ
^F	006	6	ACK
^G	007	7	BEL
^H	010	8	BS
\t	011	9	HT
\n	012	10	LF
^K	013	11	VT
\f	014	12	FF
\r	015	13	CR
^N	016	14	SO
^O	017	15	SI
^P	020	16	SLE
^Q	021	17	SC1
^R	022	18	DC2
^S	023	19	DC3
^T	024	20	DC4
^U	025	21	NAK
^V	026	22	SYN
^W	027	23	ETB
^X	030	24	CAN

^Y	031	25	EM
^Z	032	26	SB
^[033	27	ESC
\034	034	28	FS
^]	035	29	GS
^_	037	31	US
	040	32	SP
\"	042	34	"
\\	134	92	\
^^	136	94	^
^?	177	127	DEL

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

Handling Winsock application data flow

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

Original code

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

```

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

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

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

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

```

QALoad 5.02

```
#ifndef NULL
#define NULL 0
#endif

int rrobot_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);

#ifdef NULL
#define NULL 0
#endif

int rrobot_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");

```

QALoad 5.02

```
////////////////////////////////////
// Finally, substitute the Session ID received from
// the server with the one coded in the script.
////////////////////////////////////
sprintf(SendBuffer, "%s:^B^@^@^B^@^@^A^@^@^", Buffer);
DO_WSK_Send(S1, SendBuffer);

/* 34 bytes */

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

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

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

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
RR_printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
```

Modifying QALoad's functions to incorporate dynamic data

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

Scenario 1:

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

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

Example

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

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

...

BEGIN_TRANSACTION();
GET_DATA();

...

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



```
strcpy(temp,VARDATA(1));
OctalToChar(temp); //used to convert octal strings
                  //to their binary format

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

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

Scenario 2:

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

Example

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

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

...

BEGIN_TRANSACTION();

...

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

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

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

/* 2 bytes: ok */

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

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

Saving server replies

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

Using the `Response()` and `ResponseLength()` commands

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

Example

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

```

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

...

BEGIN_TRANSACTION();

...

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

/* 21 bytes: You are now connected */
DO_WSK_Expect(S1, "d");

// Used to store the data that was received by the
// DO_WSK_Expect
temp = Response();

// Used to get the size of the response that was received
// so far
x = ResponseLength();

/* The line below will print the length of the response and the actual response */
RR_printf("length = %d, and response= %s",x, temp);
DO_WSK_Closesocket(S1);

```

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

Using the DO_WSK_Recv() command

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

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

Example

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

```

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

...

BEGIN_TRANSACTION();


...

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

```

```
/* 21 bytes: You are now connected */
memset(temp, '\0', 45);
DO_WSK_Recv(S1, temp, 45, 0, &size);
RR_printf("size=%d string=%s", size, temp);
DO_WSK_Closesocket(S1);
```

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

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

Parsing server replies for values

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

Scenario 1:

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

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

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

```
/* Declare Variables */
char temp[35];
int size = 0;
...
BEGIN_TRANSACTION();
...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 23 bytes: sessionid=1234567890abc */
DO_WSK_Expect(S1, "c");
SkipExpr("sessionid=");
size=ScanExpr(".*abc" , temp);
RR_printf("length = %d string = %s", size, temp);
DO_WSK_Closesocket(S1);
```

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

Scenario 2:

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

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

Example

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

```
/* Declare Variables */
char temp[15];
...
BEGIN_TRANSACTION();
...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 16 bytes: xxx0123456789yyy */
memset(temp, '\0', 15);
DO_WSK_Expect(S1, "yyy");
ScanSkip(3);
ScanString(10, temp);
RR_printf("string=%s", temp);
DO_WSK_Closesocket(S1);
```

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

WWW

Simulating variable IP addresses

While QALoad can simulate multiple virtual users from a single system, it generally does so using a single source IP address. In most testing situations this isn’t a problem, but with a small set of HTTP-based applications, it may not be the best way to simulate real-life activity. For QALoad Player machines with more than one static IP address, QALoad can direct each virtual user to use a different source IP address. To accomplish this, a local datapool file containing a list of local static IP addresses must be created on each QALoad Player machine. When you enable IP spoofing in the QALoad Conductor, the QALoad Conductor instructs each QALoad Player to create the appropriate datapool file at run time. The QALoad Player 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

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

This option inserts the `DO_IPSpoofEnable` command directly in the script during conversion, before the first `DO_Http` or `DO_Https` command.

Creating a Datapool of IP Addresses


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

To create a datapool of IP addresses:

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

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

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

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

Handling error messages from the Web server

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

Handling error messages with response codes

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

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

Example

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

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

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

Handling error messages returned in an HTML page

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

Using DO_VerifyDocTitle to verify page requests

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

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

Example

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

Searching response text for error messages

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

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

Example

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

Simulating CGI requests

The following topics describe strategies for simulating CGI requests:

- [CGI parameter encoding](#)
- [CGI Get requests](#)
- [CGI Post requests](#)
- [CGI forms](#)

Simulating JavaScript

JavaScript is handled by the following process:

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

Supported objects

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

Supported properties

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

Evaluation errors

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

Example Web page

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

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

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

function scrollit_r21(seed)
{
var m1 = " Welcome to Compuware's QALoad homepage.";
var m2 = " Glad to see you.";
var m3 = " Thanks for coming. ";
var msg = m1 + m2 + m3;
var out = " ";
var c = 1;

if (seed > 100) {
seed--;
var cmd="scrollit_r21(" + 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_r21(" + 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_r21(" + seed + ")";
window.status=out;
timerTwo=window.setTimeout(cmd,100);
}

else {
window.status=" ";
timerTwo = window.setTimeout("scrollit_r21(100)", 75);
}
}
}

</script>

<BODY onLoad="timerONE=window.setTimeout('scrollit_r21(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

QALoad 5.02

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

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

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

Executing a Visual Basic script

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

Executing a Java applet

Java applets are handled by the following process:

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

Example Web page

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

```
<HTML>  
<HEAD>  
<TITLE>Java Example</TITLE></HEAD>  
<BODY>  
  
<center><h2>Java Applet Example</h2><hr>  
  
<applet code="LScrollText.class" width="500" height="20" >  
<PARAM NAME="MESSAGE" VALUE="Scrolling Text created by Java Applet... >>Click here to  
Download<< Use it FREE">  
<PARAM NAME="FONTHEIGHT" VALUE="14">  
<PARAM NAME="SPEED" VALUE="2">  
<PARAM NAME="PIXELS" VALUE="1">  
<PARAM NAME="FONTCOLOR" VALUE="0000FF">  
<PARAM NAME="BACKCOLOR" VALUE="FFFF00">  
<PARAM NAME="TARGET" VALUE="lscrolltext.zip">  
</applet>  
  
<br><br><br>  
A scrolling message, with custom colors, font size, speed, and target URL.<br>  
The source (.ZIP) file can be downloaded by clicking the associated area in text window.  
<br><br><br><hr>
```



```

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

</APPLET>

<br><br><br>

```

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

<hr>

```

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

```

Example script

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

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

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

```

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

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

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

...
...
END_TRANSACTION();

```

Simulating frames

Frames are handled by the following process:

QALoad 5.02

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

Example Web page

The following Web page contains four frames.

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

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

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

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

</HTML>
```

Example script

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

- ! A DO_Http call to retrieve the main page.

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

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

...
...

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

...
...

END_TRANSACTION();
```

Simulating cookies

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

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

Example Web page

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

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

```

<html>
...
The cookies for this site are:<br><br>
<B>SaneID=172.22.24.180-4728804960004; SITESERVER=ID=f0544199a6c5970a7d087775f83b23af
</B><P>
<b>Next cookie for this URL will be : 1</b><br>
<br>RELOAD PAGE TO INCREMENT COUNTER<br><br><A HREF=http://www.host.com/index.htm>Return to
previous homepage.</A>

```

Example script when Dynamic Cookie Handling is turned on

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

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

```

BEGIN_TRANSACTION();
DO_DynamicCookieHandling(TRUE);
...
...
/* Request: 1 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n\r\n");
/* Request: 2 */
DO_Http("GET http://www.host.com/cgi-bin/cookies5.pl"
        "HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();

```

Example script when Dynamic Cookie Handling is turned off

The example script features the following elements:

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

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

DO_GetCookieFromReply stores the cookie values in variables, which the script then passes back to subsequent CGI requests using the DO_SetValue command.

```

int i;
char *Cookie[4];
...
...
for(i=0;i<4;i++)
Cookie[i]=NULL;
DO_InitHttp(s_info);
...
...

```

QALoad 5.02

```
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 QALoad Script Development Workbench disables browser caching while recording, which means a page is always retrieved.

Requesting password-protected directories

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

Example script

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

The example script features the following elements:

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

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

```

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

...
...
END_TRANSACTION();

```

Example script

QALoad also handles Windows Domain Authentication (NTLM).

The example script features the following elements:

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

```

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

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

...
...
END_TRANSACTION();

```

Using the WWW Convert Options dialog box

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

- [Form fields as comments](#)
- [Anchors as comments](#)
- [Client image maps as comments](#)
- [Debug comments](#)
- [Document title verification](#)
- [Baud rate](#)
- [Refresh timeout](#)
- [Encode DBCS characters](#)
- [Enable Visual Navigator](#)

Advanced options:

- [Cache](#)
- [Dynamic redirect handling](#)
- [Dynamic cookie handling](#)
- [Automatically process subrequests](#)
- [Persistent connections during replay](#)
- [Reuse SSL session ID](#)
- [Max concurrent connections](#)
- [Max connection retries](#)
- [Server response timeout](#)
- [HTTP version detection](#)
- [ActiveData](#)
- [IP spoofing](#)
- [Streaming media](#)
- [Hostnames as IP addresses](#)


[Strip all cookies from request](#)
[Traffic filters](#)

Testing a script

Script validation

Before adding a script to a load test, validate it to ensure it will run without problems. The following procedure is only valid for Win32 scripts. To validate a UNIX script, see [Validating a UNIX script](#).

To validate a Win32 script:

1. With a session open in the QALoad Script Development Workbench, click **Options>Workbench** to configure the Script Development Workbench and Player for validation.
2. Select the **Automatically Recompile** check box if you want QALoad to compile a script before attempting to validate it. QALoad will list any compilation errors in the editor after compiling.
3. Select the **Only Display Player Output on Script Failure** check box to only view Player messages upon script failure.
4. Type a value in the **Wait up to** field that the QALoad Script Development Workbench should wait for a script to execute before timing out.
5. In the Player Settings area, select the **Abort on Error** check box for QALoad to stop script execution upon encountering an error.
6. Select the **Debug Data** check box for the script to display a debug message indicating which command the script is executing.
7. In the Run As area, indicate whether the transaction should be run as thread- or process-based.
 **Note:** Oracle Forms Server, Citrix, Java, Uniface, and Tuxedo scripts are limited to process-based validation only.
8. In the **Number of users** field, type a number of virtual users to run this script for validation. The default is 1.
9. Enter a value in the **Transactions** field. For validation, Compuware recommends that you accept the default value of 1 transaction.
10. In the Sleep Factor % field, type the percentage of each **DO_SLEEP** (pause in the script) to maintain. For validation, you may not need to run every pause in the script at its full length. The value can be a percentage between 0 and 100. The default is 0.
11. Click **OK** to save your changes.
12. In the Workspace Pane, click the **Scripts** tab.
13. Double-click on the appropriate script name to open the script.
14. From the **Session** menu, choose **Validate Script**.

You will receive a confirmation message if the script executes successfully. If it does not execute successfully, a Validation file (.val) will open in the editor to help you identify errors.

Debugging a script

Log files

Log files can be generated for Oracle, Oracle Forms Server, Citrix, WWW, Uniface, DB2, ODBC, SAP, and Winsock scripts only.

If you encountered errors while validating or testing a script, you can view any log files generated during the test from the Script Development Workbench's LogFiles tab. Log files are generated during a test if you set debug options while setting up your test in the Conductor. Each virtual user for which you enabled

Logfile Generation will have created a file containing information about its performance. When a test finishes running, all log files are saved in the directory \Program Files\Compuware\QALoad\LogFiles. Log files are named <scriptname>_<middleware>_vu<AbsoluteVirtualUserNumber>.<ext>, where:

- ! <scriptname> is the name of the script the virtual user ran
- ! <middleware> is the name of your middleware application
- ! <AbsoluteVirtualUserNumber> is the identification number assigned to the virtual user
- ! <.ext> is the file extension, dependent upon which middleware application you are testing. File extensions are listed in the following table:

Middleware	File Extension
Oracle WWW Citrix	.rip — A log file generated by a failed Player. At the end of a test, all .rip files are sent from the Players to the \QALoad\LogFiles directory and added to the merged timing file for your analysis.
Uniface WWW	.cap — A standard log file containing information about all statements executed during a test.
Citrix DB2 ODBC Oracle Oracle Forms Server SAP Winsock WWW	.log — A standard log file containing information about all statements executed during a test.

Verifying script checkpoints

You can quickly verify the syntax of the checkpoint commands BeginCheckpoint() and EndCheckpoint() in your script every time you compile your script by setting a single option, or on-the-fly with a single menu command.

Automatically, every time you compile a script:

1. From the Script Development Workbench's main menu, click **Options>Workbench**.
2. On the Configure Script Development Workbench dialog box, click the **Compiler Settings** tab.
3. Select the **Verify Checkpoints** option .
4. Click **OK**.

Every time you compile your script, the Script Development Workbench will verify the syntax of your checkpoint statements, and ensure the parameters passed in each pair match. If any errors are encountered, an error message will display in the Output pane. You can click on any error line to go directly to that line in the script.

Manually, for the open script only:

With your script open in the Workbook pane, click **Session>Verify Checkpoints**.


The Script Development Workbench will verify the syntax of your checkpoint statements, and ensure the parameters passed in each pair match. If any errors are encountered, an error message will display in the Output pane. You can click on any error line to go directly to that line in the script.

Using EasyScript

ADO

ADO Recording Options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you select the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

 **Note:** If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the **Capture Initialization Phase** check box on the [Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box](#).

Automatic: Select this option for QALoad to automatically start your application for recording, allowing you to record early application startup activity. This is the recommended method of recording calls, because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. Choose this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.


Command Line: If you chose Automatic Program Startup, enter the command line of your application. You can also use the browse button to locate your application.

Working Directory: Enter the working directory of your application.

ADO Conversion Options

Field Retrieval: Select this option to include all instances of the ADO CARecordset->GetFields (represented in a QALoad script as ADO_Recordset(#)->GetFields(ADOFieldSet[#]); in the converted script.

Clearing this field removes a number of different elements from the script that may not be necessary for playback because they are processed on the client side rather than the server side. Removing them can greatly decrease the size of your script without affecting your load test results.

 **Caution:** Clear this option only if you are certain that ADO_Recordset(#)->GetFields(ADOFieldSet[#]); is not integral to your script.

For more information about this option, see [Using the Field Retrieval option](#).

ADO Method Reference

QALoad provides descriptions and examples of the various methods that are available for an ADO script. For details, refer to the Language Reference Help section for [ADO](#).

Citrix

Overview

Use QALoad's Citrix middleware to load test systems that run Citrix MetaFrame or Citrix MetaFrame XP.

What is Citrix?

Citrix middleware is a communication layer that provides remote access to Windows systems. The remote system appears in a window on the local system.

Connecting to the remote system

Once you have connected to a machine that is running the MetaFrame server, you can log in to the remote system and then run applications. Alternatively, you can specify an application in addition to a user name and password, which provides access only to the specified application and minimizes user input that is necessary to access the application under test.

Testing in load-balanced environments

If you are testing an environment that includes a server farm, you can use Citrix ICA files to support this type of configuration. Specify the ICA file on the [Citrix Record Options](#) dialog box. ICA files are also necessary for encryption, Published Applications, and Published Desktops. ICA files are generated on the MetaFrame server and can be obtained from your MetaFrame administrator. For more information about using ICA files, see [Using ICA files](#).

What do you want to do?

[Record a Citrix session](#)

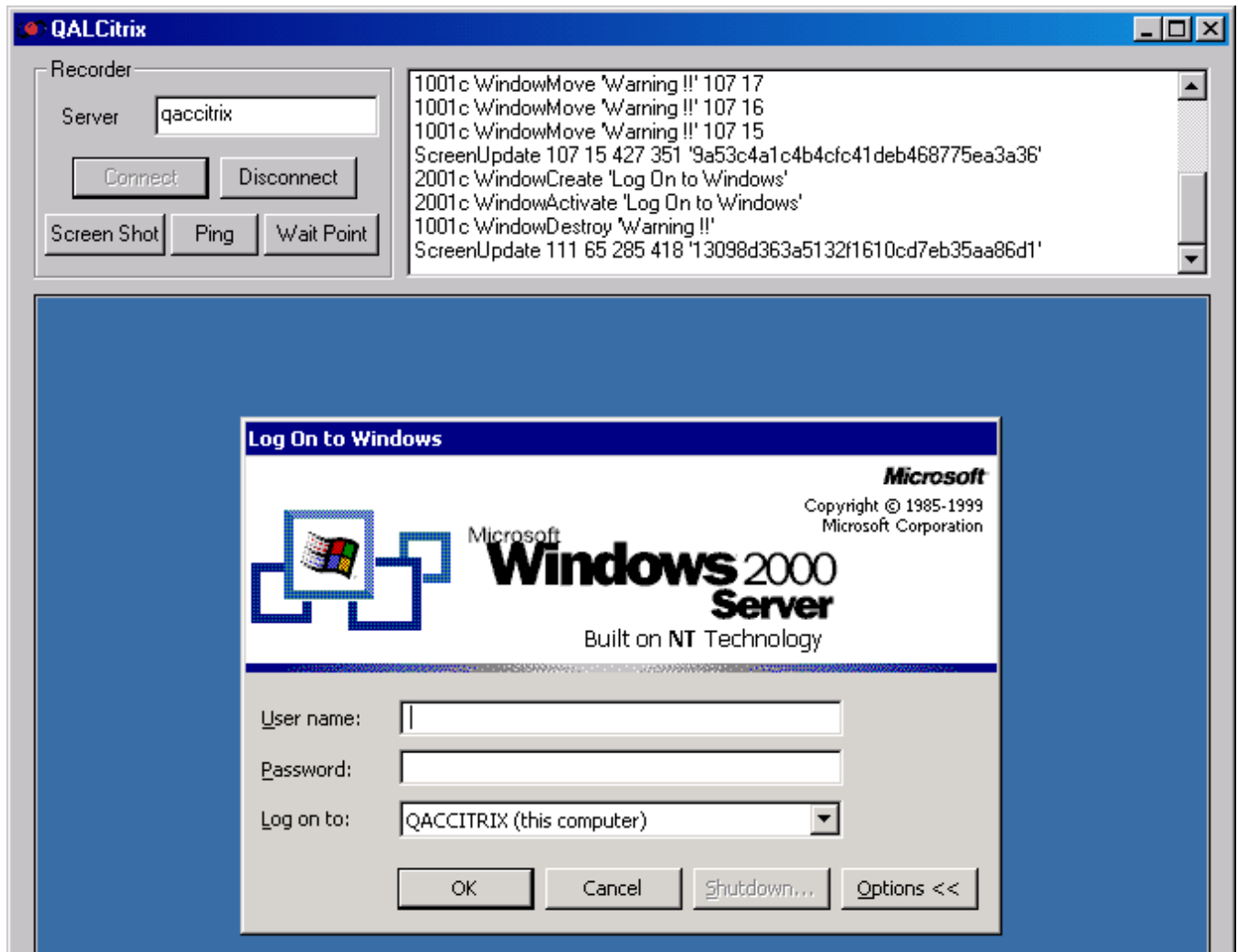
[Set recording options](#)

[Set conversion options](#)

[Recording a Citrix session](#)

To begin recording a Citrix session, click the Record button on the [Session](#) toolbar. (If you have not already chosen Citrix as the session type, click the [Citrix Session](#) button to activate a new Citrix session.) The Citrix capture application appears, as shown in the following image.

Click the three sections of the image to learn more about the fields and the information that is displayed in each area.



Citrix recording options

To set recording options, choose Record... from the Options menu in the Script Development Workbench. Set the following options for recording Citrix applications:

Server Information

Server: Type the name or address of the server machine for automatic connections. To connect to a server manually, do not enter a value in this field.

Username: Type the user name for the server machine that was specified in the Server field.

Password: Type the password for the user name that was specified in the Username field.

Domain: Type a domain name that applies to the user name and password that were specified in the Username and Password fields. Specifying a domain is optional. To ensure that the specified user name is logged on to the server instead of a domain, type the server name in this field.


Application Information

Select whether to automatically start an application after logging on. Choose Autostart to launch the application specified in the Application field, Custom to refer to the [ICA file](#) for application information, or None to disable automatic application startup.

Application: Type the path and file name of an application to start upon a successful log on.

Directory: Type the working directory for the application that was specified in the Application field. Specifying a working directory is optional even if you have specified an application in the Application field.

ICA File: Type the name of a Citrix ICA file. To enable this field, click the Custom option. Specify a URL or a file name without a path. This file, which is located on the MetaFrame server, contains configuration options for the Citrix client. For more information about ICA files, consult your Citrix administrator.

 **Notes:** To test Published Applications and load-balanced environments (server farms), you must specify an ICA file.

To validate a script on the same machine on which it was captured, you must copy the ICA file to the `QALoad\BinaryFiles` directory. To use the ICA file on remote Player machines, the ICA file should be specified as an attached file in the External Data column of the [Script Assignment tab](#) in the Conductor.

Connection Information

Port: Type the port number over which Citrix ICA traffic travels. This port is also the port on which the MetaFrame server is listening. The default port is 1494.


Resolution: Choose a window size for the Citrix connection.

Using ICA files

ICA files, which are generated on the MetaFrame server, contain configuration options for Citrix. You can specify an ICA file on the [Citrix Record Options](#) dialog box.

ICA files are specified in the script with the `CtxSetICAFile` command. If an ICA file is specified, the call is generated with an unqualified file name. For example:

```
CtxSetICAFile("customapp.ica");
```

 **Note:** The file name is not fully-qualified because the file may not exist in the same location among the remote Player machines.

To validate the script on the same machine on which it was captured, copy the ICA file to the `QALoad\BinaryFiles` directory.

To use the ICA file on remote Player machines, the ICA file should be specified as an attached file in the External Data column of the [Script Assignment tab](#) in the Conductor.

Citrix conversion options

To set conversion options, choose Convert... from the Options menu in the Script Development Workbench.

Set the following options for Citrix conversions:

General

Replay Output Mode: Choose the playback mode to use during replay.

- ! Normal mode is normal headless replay.
- ! Renderless mode maximizes the number of possible client sessions (and minimizes CPU usage) by discarding all graphic data immediately after receipt. However, you cannot take snapshots in renderless mode.
- ! Windowless mode reduces CPU usage by allowing the client to skip drawing the screen image. Rendering still exists off-screen, which makes session snapshots possible.

Enable Counters: Select to enable the middleware counters that are built in to `Connect()`, `Disconnect()`, and `Ping()` calls. Enabling these counters can affect load test performance.

Keyboard/Mouse Input

Combine consecutive key characters into a string: Select to combine consecutive ASCII character key actions. This option combines individual calls to type characters into one call for the entire string.

Convert consecutive mouse commands into points/clicks: Select to consolidate consecutive MouseMove commands into a Point() command and to convert matching MouseDown/MouseUp command pairs into Click() commands.

Timeout Values

Connect: Type the number of seconds to wait for the connection to complete.

Ping: Type the number of seconds to wait for the Ping() call to return results.

Disconnect: Type the number of seconds to wait for the disconnection to complete.

Wait Point: Type the number of seconds to wait for wait points to complete.

Window events: Type the number of seconds to wait for window creation and destruction events to occur.

Restore Defaults: Click to set all timeout options to the default settings.

Window

Enable verification: Select to enable window verification. When this option is enabled, a window must be active for an action to be issued. Also choose the number of times to re-try for verification if an active window is not found, and the number of milliseconds to wait between each try.

Enable Wildcard Title Match: Select to enable wildcard comparisons for matching Citrix window creation events. For more information about wildcard comparisons, see [Handling dynamic window titles](#).

[Citrix command reference](#)

QALoad provides descriptions and examples of the various commands available for a Citrix script. For details, refer to the Language Reference Help section for [Citrix](#).

[Advanced scripting techniques for Citrix](#)

Handling dynamic windows

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

Using the CtxWaitForScreenUpdate command

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

Handling dynamic window titles

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

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

2. **Verify whether there is an issue with dynamic window titles.**
When a script fails on validation because the run time window title is different than the expected window title from the recording, it is likely that you are dealing with a dynamic title issue that can be handled by this scripting technique. In this case, the script fails on the `CtxWaitForWindowCreate` call.
3. **Identify a match “pattern” for the dynamic window title.**
Note the error message that is returned during validation (or replay). The message indicates the expected window title versus the window title from script playback. Examine the differences in the window titles to create a “match pattern” that recognizes the window title, while ignoring other windows. A match pattern can be a simple substring of the window title or a pattern string using wildcard characters such as `?` (to match any single character) or `*` (to match any number of characters). The examples below illustrate the different match patterns.
4. **Insert a `CtxSetWindowMatchTitle` command prior to the `CtxWaitForWindowCreate` call for the dynamic window.**
When adding the `SetWindowMatchTitle` command, ensure that the first parameter contains the correct window object and the second parameter contains the match string in double-quotes.
5. **Validate the script to ensure the `CtxWaitForWindowCreate` command recognizes the dynamic window name.**
Run the revised script through validation to ensure that the script succeeds. If the script does not validate successfully, go to step 3 to determine if the match pattern is correct.

Example 1: Using a substring match

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

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

Example 2: Using a wildcard match with the `*` character

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

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

Example 3: Using a wildcard match with the `?` character

In this example, the `Random Value` 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 “`Random Value:` ”, followed by single digit:

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

Handling dynamic windows that require user interaction

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

To work around this issue, follow these steps:

1. Capture a session in which the dynamic window appears and the user performs the action to dismiss the window. This may require multiple attempts to capture the window. Once this is captured in a recording, save the script as a temporary script.

2. If the window did not appear in the primary script, extract the code snippet from the temporary script that acts on the dynamic window and insert it into the real script. The code usually consists of a CtxPoint command and a CtxClick command for this window. Insert the commands after the CtxWaitForWindowCreate command for the dynamic window. In addition, extract and insert the Citrix window information object constructor call and delete call to the relevant parts of the script, changing the object name to avoid conflicting with existing window objects. Ensure that the additional code is not inserted between a CtxPoint command and a CtxClick command in the primary script.
3. Add a special CtxSetWindowMatchTitle command immediately before the CtxWaitForWindowCreate command. The first parameter of the CtxSetWindowMatchTitle command should be the correct window object. The second parameter contains a special wildcard match "*" that enables the CtxClick command to accept any window title, which ensures that even if the matching window does not appear, the command still executes successfully.
4. If the window appears in the primary script, comment out the CtxWaitForWindowCreate command for the dynamic window. Because the window itself may not appear, the CtxWaitForWindowCreate command should be commented out.
5. Validate the script. If the validation is not successful, ensure that steps 2-4 were performed correctly.

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

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

Moving the Citrix connect and disconnect outside the transaction loop

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

Recording

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

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

Scripting

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

Also comment out the DO_SetTransactionStart and DO_SetTransactionCleanUp calls.

Handling Citrix server farms

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

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

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

.
.
.
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("Orders.cpp");
CitrixInit(4);
/* Citrix replay settings */
CtxSetConnectTimeout(90);
CtxSetDisconnectTimeout(90);
CtxSetWindowTimeout(30);
CtxSetPingTimeout(20);
CtxSetWaitPointTimeout(30);
CtxSetWindowVerification(TRUE);
CtxSetDomainLoginInfo(CitrixUsername, CitrixPassword, Citrix-Domain);
CtxSetICAFile("PRD desktop.ica");
CtxSetEnableCounters(TRUE);
CtxSetWindowRetries(5, 5000);
CtxSetEnableWildcardMatching(TRUE);
SYNCHRONIZE();
```

Handling unexpected events in Citrix

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

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

Generally, to handle unexpected events, you record two scripts. The first script contains a recording of the expected events. The second script should include the unexpected events. Using the

CtxWindowEventExists and CtxScreenEventExists functions, create a conditional block of code that handles the dialogs that may appear.

Example

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

```

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

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

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

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

#include "do_citrix.h"

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

#ifdef NULL
#define NULL 0
#endif

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

    /* Citrix Window Information Objects */
    CtxWI *CWI_1 = new CtxWI(0x1001c, "Warning !!", 107, 43, 427, 351);

```



```

CtxWI *CWI_2 = new CtxWI(0x2001c, "Log On to Windows", 111, 65, 418, 285);
CtxWI *CWI_3 = new CtxWI(0x5001c, "Please wait...", 111, 112, 418, 145);
CtxWI *CWI_4 = new CtxWI(0x30030, "Citrix License Warning Notice", 125, 198,
397, 127);
CtxWI *CWI_5 = new CtxWI(0x40030, "Citrix License Warning Notice", 125, 198,
397, 127);
CtxWI *CWI_6 = new CtxWI(0x4002e, "UsrLogon.Cmd", 0, 456, 161, 25);
CtxWI *CWI_7 = new CtxWI(0x1003a, "", -2, 452, 645, 31);
CtxWI *CWI_8 = new CtxWI(0x10066, "ICA Seamless Host Agent", 0, 0, 391, 224);
CtxWI *CWI_9 = new CtxWI(0x10052, "Program Manager", 0, 0, 641, 481);
CtxWI *CWI_10 = new CtxWI(0x1008c, "", 115, 0, 405, 457);
CtxWI *CWI_11 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
CtxWI *CWI_12 = new CtxWI(0x2006a, "", 200, 186, 156, 287);
CtxWI *CWI_13 = new CtxWI(0x10138, "", 112, 116, 416, 248);
CtxWI *CWI_14 = new CtxWI(0x50036, "Microsoft Word", -4, -4, 649, 461);
CtxWI *CWI_15 = new CtxWI(0x1017e, "Open", 19, 23, 602, 387);
CtxWI *CWI_16 = new CtxWI(0x20174, "*Microsoft Word", -4, -4, 649, 461);
CtxWI *CWI_17 = new CtxWI(0x10058, "", 113, 114, 305, 26);
CtxWI *CWI_18 = new CtxWI(0x2013e, "Calculator", 66, 66, 261, 253);
CtxWI *CWI_19 = new CtxWI(0x1005a, "", 2, 49, 205, 408);
CtxWI *CWI_20 = new CtxWI(0x3006a, "Shut Down Windows", 111, 96, 418, 193);

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

SET_ABORT_FUNCTION(abort_function);

DEFINE_TRANS_TYPE("capSave11111-2.cpp");

CitrixInit(1);

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

SYNCHRONIZE();

BEGIN_TRANSACTION();

DO_SetTransactionStart();

CtxConnect(CitrixServer, CitrixOutputMode);

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

CtxWaitForWindowCreate(CWI_1, 2125);

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

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

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

.
.

```

QALoad 5.02

```
.  
  
DO_MSLEEP(63);  
// Window CWI_14 ("Microsoft Word") created 1087837397.390  
  
CtxWaitForWindowCreate(CWI_14, 141);  
  
DO_MSLEEP(78);  
CWI_14->setTitle("Document1 - Microsoft Word"); //1087837397.468  
  
// Window CWI_13 ("") destroyed 1087837397.468  
  
DO_MSLEEP(2468);  
CtxPoint(37, 50); //1087837400.218  
  
DO_MSLEEP(282);  
CtxClick(CWI_14, 203, L_BUTTON, NONE); //1087837400.421  
  
// Window CWI_15 ("Open") created 1087837400.764  
  
CtxWaitForWindowCreate(CWI_15, 344);  
  
DO_MSLEEP(1656);  
CtxPoint(132, 99); //1087837402.671  
  
DO_MSLEEP(250);  
CtxDoubleClick(CWI_15); // 1087837402.874  
  
DO_MSLEEP(109);  
  
DO_MSLEEP(1953);  
CtxPoint(247, 197); //1087837404.827  
  
// Window CWI_15 ("Open") destroyed 1087837404.827  
  
if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE,3000,CWI_16))  
BeginBlock();  
    CtxPoint(337, 265); //1087837404.905  
  
    // Window CWI_16 ("11111111 - Microsoft Word") created  
1087837404.905  
  
    CtxWaitForWindowCreate(CWI_16, 31);  
  
    // Window CWI_14 ("Document1 - Microsoft Word") destroyed  
1087837404.905  
  
    DO_MSLEEP(7547);  
    CtxPoint(628, 9); //1087837414.592  
  
    DO_MSLEEP(2141);  
    CtxClick(CWI_16, 281, L_BUTTON, NONE); //1087837414.873  
  
    DO_MSLEEP(234);  
    // Window CWI_16 ("11111111 - Microsoft Word") destroyed  
1087837415.108  
  
    CtxPoint(113, 93); //1087837418.779  
  
    // Window CWI_17 ("") created 1087837418.779  
EndBlock();
```

```

///ReadOnly Code Start

    else
    BeginBlock();

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

        CtxWaitForWindowCreate(CWI_117, 578);

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

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

        DO_MSLEEP(109);
        // Window CWI_117 ("File In Use") destroyed 1087840079.458

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

        CtxWaitForWindowCreate(CWI_118, 63);

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

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

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

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

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

    EndBlock();

///ReadOnly Code End

    DO_MSLEEP(1063);

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

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

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

    // Window CWI_7 ("") destroyed 1087837440.138

    DO_SetTransactionCleanup();

    CtxDisconnect();

```

```

        END_TRANSACTION();

        delete CWI_1; // "Warning !!"
        delete CWI_2; // "Log On to Windows"
        delete CWI_3; // "Please wait..."
        delete CWI_4; // "Citrix License Warning Notice"
        delete CWI_5; // "Citrix License Warning Notice"
        delete CWI_6; // "UsrLogon.Cmd"
        delete CWI_7; // ""
        delete CWI_8; // "ICA Seamless Host Agent"
        delete CWI_9; // "Program Manager"
        delete CWI_10; // ""
        delete CWI_11; // ""
        delete CWI_12; // ""
        delete CWI_13; // ""
        delete CWI_14; // "Microsoft Word"
        delete CWI_15; // "Open"
        delete CWI_16; // "11111111 - Microsoft Word"
        delete CWI_17; // ""
        delete CWI_18; // "Calculator"
        delete CWI_19; // ""
        delete CWI_20; // "Shut Down Windows"

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

        CitrixUninit();

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

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

    CitrixUninit();

    EXIT();
}

```

Java

Overview

QALoad does not support recording of Java scripts. Instead, Java scripts are created from script templates that you use to create a stub script that you can then edit manually. Templates are saved in the QALoad\Middlewares\Java\Templates directory. QALoad supplies four default templates. QALoad uses a token name to represent the classpath — when you create a new Java script, QALoad simply replaces the token <classnamehere> with the class/script name you assign. You can also install additional templates using the <classnamehere> token if you wish.

Following is an example of a Java template with the classname token (in bold):

```

import com.compuware.qacenter.qaload.EasyScript.*;
public class <classnamehere> implements EasyScript
{
    /** optional - Class method runs once for each script when class is loaded */
    public static void setup QALoad Test () throws Exception

```

```
{
}
```

Use the Script Development Workbench to convert a previously recorded EasyScript for RMI script to an EasyScript for Java script, or to create a new EasyScript for Java script from the provided stub scripts.

Creating a new Java script

Using my script from a previous version of QALoad

Accessing JavaDoc

QALoad provides JavaDoc for your reference. To access it from the Script Development Workbench menu, choose Help>EasyScript for Java: JavaDoc from a Java session.

Updating your scripts

To update an RMI script from a release prior to QALoad 5.0, take the following steps to convert your script to the new Java Script format.

To update RMI scripts:

1. Copy your existing script to the `QALoad\Middlewares\Java\Scripts` directory.
2. Open an EasyScript for Java session.
3. Open the script you copied to the `\Scripts` directory in step 1.
4. Create a new Java script using the template named *old format*. This template illustrates the modifications you need to make to your previous script to make it compatible with EasyScript for Java.
5. Use the comments in the new script you created using the old format template to guide you in modifying your previously created script.
6. When you are finished, save your modified script.

Creating a Java Script

To create a Java script for QALoad :

1. With a Java session open, choose **File>New** from the menu.
2. In the File area, click on the **Middleware** tree item.
3. In the Filename field, type a name for your new Java script. Note that Java file names do have special requirements, and QALoad will enforce those requirements. For example, Java file names cannot contain spaces. If you try to include a space in your file name, QALoad will give you an error prompt.
4. Click **OK**. The Create Java Script dialog box opens.
5. Under the Script field is a selection box listing all the templates available in your `\QALoad\Middlewares\Java\Templates` directory. QALoad provides four default templates. If you click on a template name, a sample is shown in the right pane. The four templates are:
 - long format — Provides all required and optional methods.
 - new class — Creates a class associated with the script.
 - old format — Shows modifications needed to run legacy scripts.

- short format — Provides only the minimum required methods.


Select the template that best suits your needs and click OK. QALoad creates a stub script by the name you designated and opens it in the Workbook pane for editing.

6. Edit your script as necessary. You can use QALoad's [Java Script Options dialog box](#) to edit some script attributes.

Oracle

Oracle recording options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

 **Note:** If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the **Capture Initialization Phase** check box on the [Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box](#).

Automatic: Select this option for QALoad to automatically start your application when recording, allowing you to record early application startup activity. This is the recommended method of capturing API calls, because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. Select this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

Command Line: If you chose Automatic Program Startup, enter the command line of your Oracle application. You can also use Browse to locate your application.

Working Directory: Enter the working directory of your Oracle application.

 **Note:** If you entered the full path in the **Command Line** field, this field is filled in automatically.

Oracle conversion options

Database Paths, Includes: Enter the location of your Oracle database includes.

Database Paths, Libraries: Enter the location of your Oracle database libraries.

Variablization (ActiveData) Enable: Select this option to enable ActiveData for Oracle during conversion. This option is enabled by default. For more information, click [ActiveData for Oracle](#).

Minimum Characters: Enter the minimum number of characters to match for auto-variablization.

PostCapture: Fetch Iteration Override: Type the number of fetch iterations allowed while recording a script to control the amount of data fetched during playback. To fetch all data, type: 1000000.

ActiveData for Oracle

ActiveData for Oracle

Oracle variablization is a powerful scripting assistant that provides automatic correlation of data values in your script (auto-variablization) and lets you use a datapool as the source of data values (manual-variablization).

Auto-variablization


When you enable auto-variablization, QALoad correlates the data values produced by the execution of recorded SQL statements and assigns a single source variable to matching bind and static variables that subsequently use the value. Auto-variablization will only target a capture file's bind variables and

embedded static data in recorded SQL statements as receivers of source variables. Source variables will be automatically generated based on the capture file's PostBind data, Fetch data, and embedded Static data in SQL statements. Source variables from PostBind data will be generated only if the PostBind data belongs to one of these OCI bind data types:

Code	OCI7 Bind Data Type	OCI8 Bind Data Type
3	SQLT_INT	SQLT_INT
4	SQLT_FLT	SQLT_FLT
68	SQLT_UIN	SQLT_UIN
1	SQLT_CHR	SQLT_CHR
5	SQLT_STR	SQLT_STR
96	SQLT_AFC	SQLT_AFC
97	SQLT_AFC	SQLT_AFC
11	SQLT_RID	SQLT_RID Not Applicable in OCI8

Source variables from Fetch data will be generated only if the Fetch data belongs to one of the above OCI datatypes or one of the following:

Code	OCI7 Fetch Data Type	OCI8 Fetch Data Type
6	SQLT_VNU	SQLT_VNU
2	SQLT_NUM	SQLT_NUM

 **Note:** Fetch data is made available in the capture file only when the Oracle Capture Option **Use Fetch data for Variablization** is selected.

Static data embedded in SQL statements will be used as source variable or receiver of a source variable only when the SQL statement states a SELECT, INSERT, UPDATE or DELETE operation. SQL statements that contain stored procedures (e.g. BEGIN...) will be excluded.

Auto-variablization occurs by default in QALoad, but you can turn it off by clearing the conversion option Variablization (ActiveData) on the Oracle Convert Options tab. If you choose to use automatic variablization, you can then use manual-variablization to change a source variable previously determined by auto-variablization to data from a local or central datapool.

Manual Variablization

Manual variablization allows you to change the source of variables identified through auto-variablization to use data from central or local datapools. You use the variablization tree-view and the options available from the tree-view to view and change source variables.

Manual variablization is limited to changing the source variables to data that was prepared from a local datapool or conductor (central) datapool. Once changed, all (but not individual) source variables may be changed back to the original source variables.

Why use ActiveData for Oracle?

- ! **To avoid duplicate key errors which can occur during playback when the data relationships hidden (implied) within a set of Oracle SQL statements are not recorded.** For example, a recorded Select SQL statement may include the Oracle nextval expression to get the next sequential unique number in the database. The returned value from the expression is used for the primary key in a subsequent Insert statement. The primary key is associated with a bind variable. The value of the bind variable is recorded and noted in the QALoad script. When the script is played back, the returned value from nextval will naturally be different from the value of the bind variable. The Insert SQL execution incurs a duplicate key error from the Oracle server.

Oracle variablization prevents this error by providing a logical relationship between the returned data from the Select statement and the data for the Insert bind variable. The data relationship is established through a source variable.

- ! **To reduce diagnostic time for playback data issues, especially when dealing with large scripts.** Using a single source variable for script variables that have the same data value reduces the amount of debugging time that would have been spent on multiple script variables. Additionally, the Compare Tool aids you in debugging data issues by highlighting SQL and data differences that could influence the load test of two similar capture files.

Variablization menu

Access the Variablization menu from the Script Development Workbench's Session menu, or by right-clicking from the variablization tree-view.

Create/ Edit a source: Opens a tree-view of your variablized statements and their sources.

Show Capture Difference: Accesses the Compare Tool, where you can choose a capture file to compare to the current capture file and have the differences in SQL statements and bind data highlighted for your comparison within the variablization tree-view.

Revariablize: Deletes all manually generated source variables and re-executes auto-variablization. Note that datapool sources may not be changed back to PostBind, Fetch, or Static data unless you select this option.

Remove all sources: Deletes all source variables from the script's .var file.

Show SQL statement: Provides a detailed view of the highlighted SQL statement. The detailed view will display associated Bind and

Column data (from the Execute statement), associated PostBind data, and associated Fetch data.

Hints: Opens the Oracle Variablization Hints online help.

Word wrap: Shows the complete SQL statement in wrapped format. This is selected by default.

Display options: Allows you to change display options to one of the following: Only statements with bind variables, Unsourced Bind statements, or Show all SQL statements (default).

The Refresh the current view option will re-draw the tree-view after a source is manually changed.

Save the Variablization VAR file: Saves any changes to the script's .var file.

Save and Convert: Saves changes to the .var file and re-converts the script.

Save and Convert As: Saves changes to the .var file and prompts you to save your script under a new name before re-converting it.

Variablize

Use this dialog box to variablize a file or to compare two similar files. The results will be displayed in a tree-view from which you can manually variablize the file or view the differences between the two files. When

you compare two files, the differences in SQL statements and bind data will be highlighted within the Variablization tree-view.

Variablize the following capture file: Lists the path and name of the currently selected capture file (.cap).

Compare and Variablize with the following file: Navigate to the capture file you'd like to compare to the currently selected capture file.

Variablize: Variablizes the file and displays the recorded SQL statements, bind variables, static variables embedded in SQL statements, data values and the sources of data values as determined by auto-variablization.

Cancel: Closes the dialog box without making any changes.

Source Details

Displays details about the source of the selected variable, and allows you to replace the source with data from a central or local datapool.

Name: Lists the name of the field in the script that was variablized.

Value: Lists the value assigned to the variablized field.

Line #: Lists the script line where the field is located.

(Default) From Postbind/ Fetch/ Static data: If this option is selected, the source of the variable was determined by auto-variablization.

Source variable name in Convert script: The name assigned to the variable by auto-variablization, or when replaced by a datapool variable.

From datapool: Select this option to change the source to a central or local datapool.

Field Number: Specify the column number in the datapool file to use as the source.

Advanced Options: Click to open the [ActiveData Advanced Source Options](#) dialog box where you can format the source before using it, if necessary.

Display values matched by auto-variablization: In this area, click the appropriate button to determine which values to display: Sources, Matching values, or Matching names and values.

Match exact: Select if the source must be an exact match, or de-select to use the source for a sub-string search.

Update Source: Click to update the variable source according to the settings on this dialog.

Update ALL: Click to use the newly created source variable for all items in the list area.

Delete Source: Click to delete the variable and all its references from the tree-view.

Quit: Click to cancel without saving any changes.

Comparing files

The Oracle Variablization Compare Tool compares two similar capture files and highlights the differences in SQL statements and bind data and highlights them in the Variablization Tree View.

Why use the Compare Tool?

The Compare Tool can help you debug data issues in your transaction that may cause load test problems, especially in large scripts. With the differences highlighted in a window display, you can quickly determine if manual variablization is warranted for specific variables. Manual variablization can help you work around data issues that influence load tests.

To use the Compare Tool:

1. In the Workspace pane, right-click on the first capture file you want to compare and select **Variablize** from the shortcut menu. The Variablize dialog box opens, displaying the path and name of the selected file.
2. Select the **Compare and Variablize with the following file** check box, and then navigate to the capture file you wish to compare against the first selected file.
3. Click the **Variablize** button. A new tab opens in the Script Development Workbench, presenting a tree-view of the data. Differences in SQL statements and bind data are highlighted.
4. View differing values by clicking on a highlighted bind variable or SQL statement. The Show Capture Difference window will open, listing the value used in each file.
5. If you don't need to change the data, click **OK** to be returned to the tree-view. If you need to change the source of a bind item to a datapool variable, click **Go to Source Display**. The Source Details (for bind data) window or Show SQL Statement (for SQL statements) window opens.
6. Change the source of any variables to call datapool items.
7. Save the .var file and convert your capture file to build an updated script by right-clicking and selecting **Save and Convert** or **Save and Convert As**.

Setting up QALoad to run Oracle scripts on UNIX

After installing the QALoad UNIX Player and utilities, you should ensure that the following environment variables are set prior to starting the Player Agent (loadagent):

Platform	Environment Variable	Value
All Platforms:	ORACLE_HOME	<path>/oracle/product/<version>
	TNS_ADMIN	<location of config files>
	ORACLE_SID	<oracle instance name>
AIX:	LIBPATH	<playerdir>/lib:<ORACLE_HOME>/lib
HP-UX:	SHLIB_PATH	<playerdir>/lib:<ORACLE_HOME>/lib
Linux:	LD_LIBRARY_PATH	<playerdir>/lib:<ORACLE_HOME>/lib
Solaris:	LD_LIBRARY_PATH	<playerdir>/lib:<ORACLE_HOME>/lib

Setting environment variables on UNIX systems depends on your login shell. For example:

- ! For ksh: `export ORACLE_HOME=/oracle/product/8.1.6`
- ! For csh: `setenv ORACLE_HOME /oracle/product/8.1.6`

The ORACLE_HOME environment variable points to the directory where the Oracle workstation software has been installed. The TNS_ADMIN environment variable should point to the location of the client and/or server config files. ORACLE_SID should be set to the name of the Oracle instance. For each UNIX platform, update the appropriate library path variable to include the library directory for the particular version of Oracle.

Scripts will automatically be downloaded to the Player machines by the Conductor and compiled, if necessary, at test execution time.

During the automatic script download and compile, if a script compile error occurs, a scriptname.err file will be generated in the scripts directory.

To compile a script by hand, use the Rmake command. The syntax is as follows:

```
Rmake <scriptdir>/<scriptname>
```

or

```
Rmake <scriptdir>/<scriptname>
```

Oracle command reference

QALoad provides descriptions and examples of the various commands available for an Oracle script. For details, refer to the Language Reference Help section for [Oracle 7](#), [Oracle 7/8](#), or [Oracle 8](#).

Oracle Forms Server


Recording from Oracle Forms Server

QALoad supports recording Oracle Forms 9i and patched 6i (versions 6.0.8.14 and up) applications in HTTP mode, Forms 4.5, 6.0, 6i, and 9i applications in socket mode, and SSL-enabled Oracle Forms 9i applications. These recording types are described briefly below.

Oracle Forms scripts from QALoad versions 5.1 and earlier are in Java; scripts recorded in later versions are in C++.

Recording 9i and Forms 9i (HTTP mode)

Oracle 9iAS uses HTTP tunneling to send Forms data across the firewall as normal Internet traffic. To record Oracle 9i and Oracle Forms 9i in HTTP mode, select 9i on the Record Options dialog box before you start recording your application. Once you start recording, you must specify the Oracle Application Server name and port on the initial browser page, and then click the link to your Forms 9i application.

 **Note:** When using server-side recording, you must perform steps to configure the server. See [Using server-side recording](#) for more information.

Recording 9i (SSL mode)

To record an Oracle 9i application in SSL mode, select Forms Version 9i on the Record Options dialog box, and type the Jinitiator certDB file that the 9i application uses. The certDB file verifies the SSL Certificate Authority on the client side prior to the Forms connection. Once you start recording, specify the Oracle Application server name and port on the initial browser page, and click the link to the Forms 9i application.

 **Note:** SSL mode is not available with server-side 9i recording.

Recording Forms 4.5, 6.0, or 6i (socket mode)

For socket-mode recording, QALoad must start your application for you through your browser. Before recording, enter the URL of the Forms applet page in the URL field, and the Forms Server port in the Port field. If you leave the Port field blank, or enter an incorrect port number, your recording will only result in an empty capture file. You may leave the URL field blank, but will be prompted for the Forms applet page on the initial browser page. From the applet page, click the link to your Forms application. QALoad will take over recording at this point.

Using server-side recording

For Forms applications running in Oracle 9iAS, you can use server-side recording, which avoids issues that can be encountered with some server configurations. Server-side recording creates a script recording by using the Forms server's own capabilities to record transactions.

To enable server-side recording, select the 9i Server-Side Recording check box on the [Oracle Forms Server Record Options dialog box](#) and provide the URL of the ListenerServlet in the ListenerServlet field.

Server setup

Before recording a script using server-side recording, you must first modify the server configuration so that QALoad can communicate with the server properly. Once the server modifications are complete, recording and playback are the same as for standard OFS scripts.

To prepare the Forms server for server-side recording:

1. Copy the `ofsmmessage.jar` file from the `\QALoad\Classes` directory of the QALoad installation on the client machine to the `\FORMS90\JAVA` directory of the Oracle 9i Application Server installation on the server machine.
2. Add a new section of configuration parameters to the `formsweb.cfg` file in the `\FORMS90\server` directory of the Oracle 9i Application Server installation. Use the following format, substituting your own information for the items in boldface type:

```
[MsgBlk]
form=test1.fmx
userid=scott/tiger@iasdb
archive_jini=f90all_jinit.jar,OfsMessage.jar
archive_ie=f90all.cab,OfsMessage.jar
archive=f90all.jar,OfsMessage.jar
formsMessageListener=oracle.forms.iserver.MessageListener
recordFileName=c:\temp\is
```

3. when you begin to record, append a config parameter on the initial browser page's URL, as shown in boldface type in the following sample URL:
`http://ntsap45b:7779/forms90/190servlet?config=MsgBlk`

Oracle Forms Server recording options

QALoad records through your default browser.

Connections

Forms version: Select your Oracle Forms Server version. This parameter must match the version of WebForms you are using or you will not be able to record. If you use the Oracle Applications suite of products, use the following table to determine the Forms version:

Applications Version	Forms Version
10	4.5
11	4.5
11i	6i

Enable server-side recording: Select to enable Oracle Forms 9i server-side recording.

Listener servlet URL: Type the URL of the listener servlet that is used by the application under test.

Use SSL: Select this option if the application uses Forms Version 9i and the application is SSL-enabled.

Certificate file: For SSL mode only. Type the name of, or browse for, the Oracle Jinitiator's certDB file that is located on the client machine. The certDB file is used by Oracle 9i to verify the SSL Certificate Authority on the client side prior to the Forms connection.

Socket URL: For socket connections with Oracle Forms versions 6i, 6.0, or 4.5, enter the URL to initiate your applet. If you are recording from 9i, this field is unavailable.

Port: If your Oracle Forms version is anything other than 9i, type the port number on your Oracle Forms Server application server for QALoad to listen on. This is the same port as the Forms listener (usually 9000).

To determine the server port:

1. Start the WebForms application in the browser.
2. Once the application has started, choose the menu option **View>Source**.
3. Look in the source code for a line that resembles this:
`<PARAM NAME = "serverPort" VALUE = "9000" >`. This value is the server port.

Additional Jar Files

Override application defaults: If you need to use Java resources other than those that are your application's defaults, select the check box and then click Add to navigate to the appropriate JAR files and add them to the list.

Oracle Forms Server conversion options

General

Send heartbeat every [n] minutes: Type the number of minutes between each Forms 6i heartbeat message. Compuware recommends using a value of 4 or more to prevent socket usage issues.

Simulate an Oracle Applications 11i Login: Select to simulate an Oracle Applications 11i login via a Personal Home Page. Then complete the following three fields:

HomePage URL: Type the location (URL) of the Personal Home Page.

Userid: Type the user name to be used on the Personal Home Page.

Password: Type the password for the user name that is specified above in the Userid field.

CPP scripts

Stop running when server messages indicate errors: Select to check server messages for errors. If this option is selected, the script stops running if errors are encountered. If this option is not selected, the script ignores errors and continues.

Stop only if server message matches specified string: Select to allow script failure only if the specified server message is received. This option is available only if the Stop running when server messages indicate errors is also selected.

Message: Type a string to match against server messages, and select one of the following match types.

contains: Match based on the specified characters appearing anywhere in the message.

is exactly: Match based on the specified complete error message name.

begins with: Match based on the specified characters appearing at the beginning of the message.

ends with: Match based on the specified characters appearing at the end of the message.

Java scripts

Reference controls by: Select whether to reference controls in the script by name or server assigned ID number.

To use this option, the applet startup parameters must have the `record=names` option. In some applications, each time a new form or window is drawn, the server assigns it a new sequence of ID numbers. In these applications, using the originally record ID numbers can cause the script to fail when multiple transactions are executed for a virtual user.

Use of `record=names` does not consistently work for Oracle Forms Server prior to V6i. In the event you cannot use control labels, place the `BEGIN_/END_TRANSACTION` loop around the entire script.

Output a warning if name is not found: This check box is enabled when the Use "string" control labels check box is selected. The default behavior of the Script Development Workbench is to output a warning when a control label is not found. These warnings will appear as comments in the script. Selecting this option disables these warnings.

Reduce script lines: Select this option to reduce the number of script lines in socket mode capture files. This option makes it easier to read/debug long scripts. If this option is selected, QALoad produces one line of script code in a situation that previously produced 3 lines: the transmission of a message containing a Forms control that has a single property. For example:

The following 3 script lines:

```
oracleFormsMsg1 = oracleForms.FormWindow( CONTROL_FormWindow006 );
oracleFormsMsg1.Add( PROPERTY_LOCATION, new java.awt.Point(0,0) );
oracleForms.XmitMsg( 4, oracleFormsMsg1 ); // Statement # = 4
```

will be reduced to one line:

```
oracleForms.FormWindow( CONTROL_FormWindow006, PROPERTY_LOCATION, new java.awt.Point(0,0) );
```

Output client messages as comments: Output raw client capture data as comments in the generated script. This data can be used to compare converted playback log files.

Output server messages as comments: Output raw server capture data as comments in the generated script. This data can be used to compare converted playback log files. Selecting this option can significantly increase the size of scripts.

Checkpoints in Oracle Forms Server scripts

EasyScript for Oracle Forms Server supports QALoad's automatic middleware checkpoint timings in both HTTP and socket modes. Default checkpoints (Begin/End Checkpoint pairs) are not supported.

Automatic checkpoints are enabled from the Conductor's Timing Options column on the Script Assignment tab and are enabled on a script-by-script basis.

At playback, automatic checkpoints are executed during the ofsSendRecv statement.

Forms validation/playback debugging options

Debug data

When the Debug Data option is enabled on the Configure Script Development Workbench dialog box for validation, or the Conductor's Debug Trace option is enabled for playback, executed script statements will be displayed. For example:

```
VU 0 : Line:90, ofsSetWindowSize( "FORMWINDOW" ,6, OFS_ENDMSG, 137, 750, 600 )
VU 0 : Line:91, ofsActivateWindow( "WINDOW_START_APP" ,11, OFS_ENDMSG, 247 )
VU 0 : Line:92, ofsShowWindow( "WINDOW_START_APP" ,11, OFS_ENDMSG, 173 )
VU 0 : Line:93, ofsFocus( "BUTTON" ,51, OFS_ENDMSG, 174 )
VU 0 : Line:94, ofsSetWindowSize( "FORMWINDOW" ,6, OFS_ENDMSG, 137, 750, 600 )
VU 0 : Line:95, ofsSendRecv( 1 ) //ClientSeqNo=2|MsgCount=6
```

Oracle Forms Server playback error codes

QALoad displays error codes during playback for specific exception messages. While debugging, refer to the table below that lists error codes and descriptions that apply to Oracle Forms Server scripts.

Most of the errors listed below are client request errors due to JVM memory issues. When the error is due to a server problem, the error message indicates a connection issue or a bad response from the server. All these errors cause playback to fail. When the error is client-related, you can work around the JVM memory issue by tweaking the Player machine's **Threads Per Player value** in QALoad Conductor. When the error is

server-related, the server is unable to handle the load. The server typically throws out connection requests, does not respond to requests, or terminates connections during playback.

Error code	Description
OFS-ERROR-001	Failed to create the replay log file.
OFS-ERROR-002	Failed while processing server detail message. Unknown control handle.
OFS-ERROR-003	Failed to send a heartbeat message.
OFS-ERROR-004	OracleAppsLogin: Error: icx_ticket not found in OracleAppsLogin, please check URL, userid, and password.
OFS-ERROR-005	Failed to set Boolean property of a RunForm object.
OFS-ERROR-006	Failed to process Boolean property of a RunForm object.
OFS-ERROR-007	Failed to set Point property of a RunForm object.
OFS-ERROR-008	Failed to process point property of a RunForm object.
OFS-ERROR-009	Failed to set Byte property of a RunForm object.
OFS-ERROR-010	Failed to process Byte property of a RunForm object.
OFS-ERROR-011	Failed to set Integer property of a RunForm object.
OFS-ERROR-012	Failed to process Integer property of a RunForm object.
OFS-ERROR-013	Failed to set String property of a RunForm object.
OFS-ERROR-014	Failed to process String property of a RunForm object.
OFS-ERROR-015	Failed to set Void property of a RunForm object.
OFS-ERROR-016	Failed to process Void property of a RunForm object.
OFS-ERROR-017	Failed to process Character property of a RunForm object.
OFS-ERROR-019	Failed to process Float property of a RunForm object.
OFS-ERROR-020	Failed to set Date property of a RunForm object.
OFS-ERROR-021	Failed to process Date property of a RunForm object.
OFS-ERROR-022	Failed to set Rectangle property of a RunForm object.
OFS-ERROR-023	Failed to process Rectangle property of a RunForm object.
OFS-ERROR-024	Failed to set ByteArray property of a RunForm object.
OFS-ERROR-025	Failed to set StringArray property of a RunForm object.

OFS-ERROR-027	Failed to process ByteArray property of a RunForm object.
OFS-ERROR-028	Failed to process StringArray property of a RunForm object.
OFS-ERROR-029	Failed to process nested message.
OFS-ERROR-030	Failed to do server-side connection.
OFS-ERROR-031	Failed to disconnect server-side connection.
OFS-ERROR-032	Failed because the server sent this error message: <message>
OFS-ERROR-033	Failed to expand the GUI control array.
OFS-ERROR-034	Failed to get the stored properties.
OFS-ERROR-035	Failed to send terminal message using server-side connection.
OFS-ERROR-036	Failed to send Forms message using server-side connection.
OFS-ERROR-037	Failed to process the Void property of a Button object.
OFS-ERROR-038	Failed to add listbox value. Bounds error. Listbox index: <index >, Listbox value: < value>
OFS-ERROR-039	Failed to find Listbox value. Value: < value>
OFS-ERROR-040	Failed to set the String property of an ErrorDialog object.
OFS-ERROR-041	Failed to process the String property of an ErrorDialog object.
OFS-ERROR-042	Failed to process the nested message of an ErrorDialog object.
OFS-ERROR-043	Failed to set the Point property of a FormWindow object.
OFS-ERROR-044	Failed to process the Point property of a FormWindow object.
OFS-ERROR-045	Failed to set the Boolean property of a FormWindow object.
OFS-ERROR-046	Failed to process the Boolean property of a FormWindow object.
OFS-ERROR-047	Failed to set the Integer property of a FormWindow object.
OFS-ERROR-048	Failed to process the Integer property of a FormWindow object.
OFS-ERROR-049	Failed to process the nested message of a FormWindow object.
OFS-ERROR-050	Failed to set the String property of a JavaContainer object.
OFS-ERROR-051	Failed to process the String property of a JavaContainer object.
OFS-ERROR-052	Failed to process the nested message of a JavaContainer object.

OFS-ERROR-053	Failed to set the String property of an LOV object.
OFS-ERROR-054	Failed to process the String property of an LOV object.
OFS-ERROR-055	Failed to set the Integer property of an LOV object.
OFS-ERROR-056	Failed to process the Integer property of an LOV object.
OFS-ERROR-057	Failed to set the Point property of an LOV object.
OFS-ERROR-058	Failed to process the Point property of an LOV object.
OFS-ERROR-059	Failed to set the Void property of an LOV object.
OFS-ERROR-060	Failed to process the Void property of an LOV object.
OFS-ERROR-061	Failed to process the nested message of an LOV object.
OFS-ERROR-062	Failed to set the String property of a LogonDialog object.
OFS-ERROR-063	Failed to process the String property of a LogonDialog object.
OFS-ERROR-064	Failed to process the nested message of a LogonDialog object.
OFS-ERROR-065	Failed to set the String property of a MenuParamDialog object.
OFS-ERROR-066	Failed to process the String property of a MenuParamDialog object.
OFS-ERROR-067	Failed to process the nested message of a MenuParamDialog object.
OFS-ERROR-068	Failed to set the String property of a PopList object.
OFS-ERROR-069	Failed to process the String property of a PopList object.
OFS-ERROR-070	Failed to set the Integer property of a PopList object.
OFS-ERROR-071	Failed to process the Integer property of a PopList object.
OFS-ERROR-072	Failed to set the Void property of a PopList object.
OFS-ERROR-073	Failed to process the Void property of a PopList object.
OFS-ERROR-074	Failed to process the nested message of a PopList object.
OFS-ERROR-075	Failed to set the String property of a TextField object.
OFS-ERROR-076	Failed to process the String property of a TextField object.
OFS-ERROR-077	Failed to set the Point property of a TextField object.
OFS-ERROR-078	Failed to process the Point property of a TextField object.

OFS-ERROR-079	Failed to set the Integer property of a TextField object.
OFS-ERROR-080	Failed to process the Integer property of a TextField object.
OFS-ERROR-081	Failed to set the Void property of a TextField object.
OFS-ERROR-082	Failed to process the Void property of a TextField object.
OFS-ERROR-083	Failed to process the nested message of a TextField object.
OFS-ERROR-084	Failed to set the Integer property of a Tree object.
OFS-ERROR-085	Failed to process the Integer property of a Tree object.
OFS-ERROR-086	Failed to set the String property of a Tree object.
OFS-ERROR-087	Failed to process the String property of a Tree object.
OFS-ERROR-088	Failed to process the nested message of a Tree object.
OFS-ERROR-089	Failed to process the nested message of a Button object.
OFS-ERROR-090	Failed to execute SSL handshake.
OFS-ERROR-091	Failed to collect Forms message.
OFS-ERROR-092	Failed to get the content length of the POST request.
OFS-ERROR-093	Failed to get new connection for a POST request.
OFS-ERROR-094	Failed to set up a new connection for an SSL-enabled POST request.
OFS-ERROR-095	Failed while posting a NULL request for a large-data response.
OFS-ERROR-096	Failed to store data from the server reply.
OFS-ERROR-097	Failed to do a re-POST request.
OFS-ERROR-098	Server reply data is invalid. If the following Java msg is null, server has terminated this Forms session. Java Msg: <message>
OFS-ERROR-099	Failed to disconnect the URL connection.
OFS-ERROR-100	Failed to connect to Forms Servlet. Check the URL and its parameters.
OFS-ERROR-101	Failed to do SSL connection to the Forms Servlet. Check the URL and its parameters.
OFS-ERROR-102	Failed to log the Forms Servlet connection.
OFS-ERROR-103	Failed to log the HTTP reply header.

OFS-ERROR-104	Failed while reading the http reply header. Server has terminated the Forms session.
OFS-ERROR-105	Failed while reading the server reply in an SSL connection.
OFS-ERROR-106	Failed to connect to the Forms Listener Servlet. Check the URL. If the URL is valid, the server is not accepting new connections.
OFS-ERROR-107	Failed to create a new URL connection for a Get request.
OFS-ERROR-108	Failed to connect to the Forms Listener Servlet in SSL mode. Check the URL. If the URL is valid, the server is not accepting new connections.
OFS-ERROR-109	Failed to log Listener Servlet connection.
OFS-ERROR-110	Failed to log initial Forms Server connection.
OFS-ERROR-111	Failed to get a URL connection for the first POST request.
OFS-ERROR-112	Server did not return the encryption keys for the first Post request. Forms Server is not accepting new connections.
OFS-ERROR-113	Failed to load loadplayerJava at startup. Library name: <libraryName>
OFS-ERROR-114	Failed to do SSL handshake.
OFS-ERROR-115	Failed to get SSL inputStream.
OFS-ERROR-117	Failed to close SSL socket.
OFS-ERROR-118	Failed to write the Forms Message.
OFS-ERROR-119	Failed to do a socket connection to the Forms Server.
OFS-ERROR-120	Server did not return the Forms encryption key during a socket connection.
OFS-ERROR-121	Failed to close the socket during a socket connection.
OFS-ERROR-122	Failed to write Forms message during a socket connection.
OFS-ERROR-123	Failed to send Forms message. Server terminated socket connection.
OFS-ERROR-124	Server reply is invalid. If Java msg is null, server has terminated this Forms session. Java Msg: <message>
OFS-ERROR-125	Failed to get the reply content. Check the URL. The URL may be invalid.
OFS-ERROR-126	JVM memory issues.
OFS-ERROR-128	LoadValue failed. LoadValue count is greater than the array length.

Oracle Forms Server method reference

QALoad provides descriptions and examples of the various methods and functions available for an Oracle Forms Server script. For details, refer to the Language Reference Help section for [Oracle Forms Server](#).

Advanced scripting techniques

Understanding the C++ script

Understanding the C++ script

Oracle Forms Server scripts are produced for Oracle Forms 4.5, 6.0, 6i, and 9i (Release 2 and later) recordings. The C++ script executes OFS-related statements by passing the statements in the script DLL to the OFSJava engine that performs the client activities and the client communication with the server. Because the C++ script statements are directly tied to corresponding methods in the OFSJava engine, modifications to the script statements are limited to changing the property parameter values through variablization.

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

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

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

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

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

Connection statements

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

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

Server-side recording connections

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

```
ofsSetServletMode(OFS_HTTP, "http://ntsap45b:7779/forms90/190servlet" );
```

HTTP connections

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

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

For example:

```
ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.9" );
ofsHTTPSetHdrProperty("Host", "ntsap45b:7779" );
ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2"
);
ofsHTTPSetHdrProperty("Connection", "Keep-alive" );
ofsHTTPConnectToFormsServlet(
"http://ntsap45b:7779/forms90/f90servlet?ifcmd=startsession" );
ofsHTTPSetListenerServletParams( "?ifcmd=getinfo&ifhost=C104444D01&ifip= "192.168.234.1"
);
ofsHTTPConnectToListenerServlet( "http://ntsap45b:7779/forms90/190servlet" );
ofsHTTPSetHdrProperty("Content-type", "application/x-www-form-urlencoded" );
ofsHTTPInitialFormsConnect();
```

HTTPS connections

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

Socket connections

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

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

Application statements

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

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

Parameters of a property statement:

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

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

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

For example:

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

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

Forms environment statements:

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

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

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

Sending messages to the server:

The ofsSendRecv statement sends the accumulated GUI controls and their properties to the Forms Server as binary data. This statement represents the point at which the client sends a Forms Terminal Message to the

server. In Oracle Forms, the client and the server must end each data block with a Terminal Message before any transmission occurs.

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

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

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

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

Getting the server reply:

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

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

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

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

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

Processing large data and delayed response scenarios:

When HTTP or HTTPS connection mode is used, Forms data is wrapped inside the HTTP reply stream. QALoad checks the HTTP header of the reply before processing the Forms data. The HTTP header sometimes indicates that the client needs to perform additional HTTP POST requests to obtain the complete Forms data. This indication occurs when the content-length of the reply is 64000 (a large data

scenario), or the content-type is "text/plain" and the HTTP header contains an "iferror:" string (a delayed response/re-post scenario). QALoad performs the necessary POST requests to obtain the complete reply data, and then translates the accumulated reply data to Forms controls and properties.

Disconnect statements

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

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

Using script logging as a debugging tool

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

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

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

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

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

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

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

Moving the OFS transaction loop

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

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

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

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

The following example shows a modified OFS transaction loop:

New position of the BEGIN_TRANSACTION statement

```

/*
NewSales
*/

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

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

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

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

```

New position of the END_TRANSACTION statement

```

/*
EndTrans
*/

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

```

QALoad 5.02

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

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

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

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

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

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

Tips:


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

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

OFS and WWW Universal sessions

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

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

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

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

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

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

To use [ofsSetICXTicket](#), you must modify the script.

To capture an Oracle Applications login with [ofsSetICXTicket](#):

1. Add the following variable declaration statements to the top of the script:


```
char *p;
char ICX_Ticket[100];
char *pTicket;
```
2. In the *.postcapweb file, find the HTTP request that returns the ICX ticket. The reply should contain a string that indicates the ICX ticket value, such as "ICX_TICKET=". Note the left and right characters that delimit the ICX ticket value. In the example in step 4, the left delimiter is "icx_ticket='" and the right delimiter is "'".
3. In the script, find the matching request line for the HTTP request.
4. After the matching HTTP request line, add the DO_GetUniqueString statement using your chosen delimiters. For example:


```
p = DO_GetUniqueString( "icx_ticket='", "'");
```
5. Add script lines that copy the extracted value into your script variables.


```
strcpy(ICX_Ticket, p);
pTicket=ICX_Ticket;
```
6. (optional) Verify the ICX ticket value.


```
RR_printf("ICX_Ticket=\"%s\"\n", ICX_Ticket);
```
7. Add the script line that passes the value of the ICX ticket to the OFS statement of ofsInitSessionCmdLine.


```
ofsSetICXTicket(&pTicket);
```

SAP

SAP 4.x

Overview

Use QALoad's SAP middleware support to load test systems that run SAP 4.0B, 4.5 or 4.6D.

What is SAP?

The SAP GUI front end is a middleware that allows user to access SAP servers from Windows. The SAP servers run various SAP business applications, such as applications for customer relationship management, human resources, and supply chain management.

Connecting to the SAP server

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

Recording scripts

To record scripts with SAP 4.x, you must [start the QALSAP application](#). The QALSAP application enables you to connect and log on to the SAP server.

SAP recording options (Versions 4.x)

Before you can successfully record transactions from a SAP-based application, you must select the Dialog (modal) option on the Help>Settings>F4 Help tab in the SAP application. You must do this for the user you are recording.

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording

Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead.

Automatic: Select this option for QALoad to automatically start your application for recording, allowing you to record early application startup activity. This is the recommended method of recording, as it takes advantage of QALoad's enhanced abilities to handle various multithreaded programming techniques.

QALoad 5.02

Command Line: Enter or browse for the path to QALSAP, then enter the startup parameter \c. For example, enter: `c:\Program Files\Compuware\QALoad\Qalsap.EXE \c`.

The following additional startup parameters are available:

Parameter	Description
\a	Version 3.1 clients only. Use this parameter in place of \c if QALSAP cannot connect to your Sapgui. QALoad will then record directly from QALSAP. For example: <code>c:\Program Files\Compuware\QALoad\Qalsap.EXE \a</code>
\m	Minimizes the Sapgui window after successfully logging in. For example: <code>c:\Program Files\Compuware\QALoad\Qalsap.EXE \c \m</code>
\r#	Where # is a number from one to five. Type \r followed by a number from one to five to indicate how many times QALoad should attempt to login to Sapgui if the first attempt times out. For example: <code>c:\Program Files\Compuware\ QALoad \Qalsap.EXE \c \r5</code>

Working Directory: Enter the working directory of your SAP application.

 **Note:** If you entered the full path in the **Command Line** field, this field is filled in automatically.

Disable SAP Enjoy During Capture: Select this check box to disable SAP Enjoy (SAP 4.6 and above) before recording. If you choose not to disable SAP Enjoy, you must manually shut down the SAP client tray application before you stop recording.

Starting QALSAP

QALSAP is the application that enables recording from QALoad for SAP 4.x scripts. To begin recording, you must start this application.

To start QALSAP:

1. From the Windows **Start** menu, choose **Run**.
2. On the **Run** dialog box, type `QALSAP`.

SAP conversion options (Versions 4.x)

Consolidate Checkpoints: Select to consolidate checkpoints with the same description into a single checkpoint. The checkpoint description is the Tcode concatenated with the first 30 characters of the screen title.

If you do not select this option, and QALoad detects a duplicate description, it will append a sequence number after the checkpoint description.

Graphical User Interface: Select to view validation in graphical mode. If selected, the script will step through validation transactions with SAPGUI running. Due to memory requirements (15-20 MB per user), select this option only if you are replaying a single user.

Animating an SAP capture file

Animating a capture file plays back the series of transactions from an SAP recording.

To animate an SAP capture file:

1. Open an SAP session in the QALoad Script Development Workbench.

2. In the Workspace Pane, click the Captures tab.
3. Select the capture file you want to animate.
4. From the **Session** menu, choose **Animate>Start**. The QALoad Script Development Workbench opens the file in QALSAP, where you can view each transaction graphically.

Viewing SAP 4.x post-test log files

If you selected the Detailed Logging option on the QALoad Script Development Workbench's SAP Conversion Options dialog box before you ran the test, QALoad automatically generated a log file for each virtual user named `saplg###.log` (where ### is the virtual user number) during the test. Each log file contains a graphical representation of the events sent to and received from the server for a particular virtual user.

You can open and delete a log file from the QALoad Script Development Workbench using the following procedure.

To open a virtual user log file:

1. With an SAP session open in the QALoad Script Development Workbench, select **File>Browse**.
2. On the **Browse** dialog box, double-click **Log Files**.
3. After QALSAP opens, select **File>Open**.
4. In the **Files of Type** field, select **Log File**. The **Browse Log Files** dialog box opens, displaying the available SAP log files.
5. Double-click on the log file you wish to open. Log files are named `saplg###.log`, where ### is the virtual user number.

QALSAP opens the selected log file. In the Line # column of the log file, each request by the client is marked by a blue "client" icon, while each response from the server is marked by a white "server" icon, as shown in the following image.

Line #	T Code	Title
44		SAP R/3
48		SAP R/3
48		SAP R/3
54		SAP R/3
54		Log Off
62		Log Off
62		

Viewing request or response details

QALSAP allows you to view detailed information about each request and response in the log file, including each logged SAP screen, and its menus, function keys, and controls.

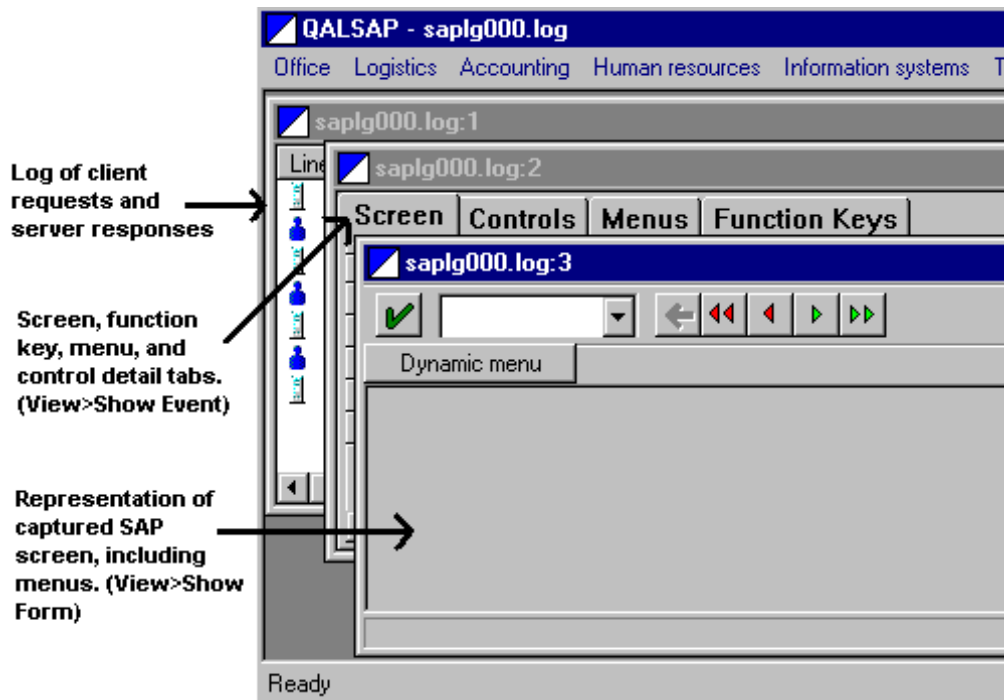
You determine which information to display by selecting one or both of the following commands from the View menu:

- ! **View>Show Event** — Opens a detail window, displaying detailed information about each logged event, including screen names, key names, tool tip text, and so on.

- ! **View>Show Form** — Opens a form window, displaying a graphic representation of the logged SAP screen and related menus.

To view a specific request or response:

1. With a log file open in QALSAP, select the appropriate command(s) from the **View** menu to determine what amount of detail to view.
2. Double-click on the line number of the request or response you want to view. Detail windows open for the selected request or response, depending on the options you set in the **View** menu. In the image below, both **Show Event** and **Show Form** were selected.
 - To view the next like response—for example, if you are viewing a client request and want to view the next client request—select View>Goto Next Response. Alternately, from the form window, click the appropriate toolbar button to view the next or previous response.
 - To view the next event in the log, select View>Goto Next Event. Alternately, from the form window, click the appropriate toolbar button to view the next or previous event.



SAP 4.x command reference

QALoad provides descriptions and examples of the various commands available for an SAP script. For details, refer to the Language Reference Help section for [SAP 4.x](#).

[SAP 6.x](#)

Overview

Use QALoad's SAP middleware to load test systems that run SAP 6.20 and 6.40.

What is SAP?

The SAP GUI front end is a middleware that allows users to access SAP servers from Windows. The SAP servers run various SAP business applications, such as applications for customer relationship management, human resources, and supply chain management.

Connecting to the SAP server

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

Configuring an SAP client for load testing

Before you can record an SAP session, you must have an SAP client that is configured to enable QALoad to access the SAP server. Configure the SAP client through the SAP Logon application.

To configure an SAP client for load testing:

1. Start the SAP Logon application. From the taskbar, click **Start>Programs>SAP Front End>SAPlogon**.

Click the **New...** button on the SAP Logon dialog box. The New Entry dialog box appears.



2. Type values in the **Description**, **Application Server**, and **System number** fields.

 **Note:** QALoad uses the value in the **Description** field to connect to the server.

3. Click **OK**. The new SAP server entry appears in the list in the SAP Logon dialog box.

SAP recording options (Versions 6.x)

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

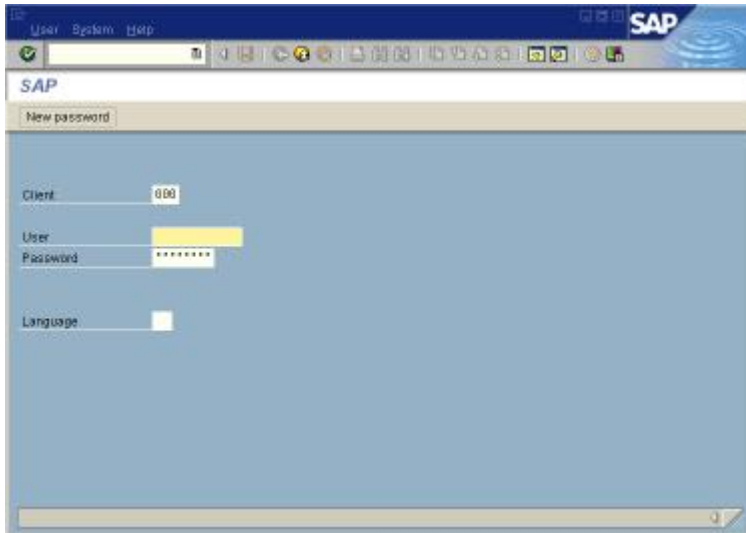
Recording an SAP session

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

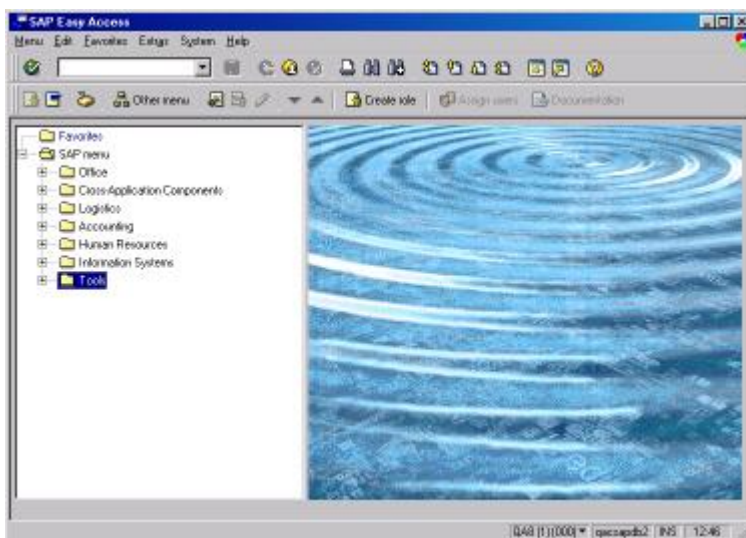
To record an SAP session:

1. In the Script Development Workbench, click the **Record** button on the Session toolbar. If you have not already chosen SAP as the session type, click the **SAP Session** button to activate a new SAP session.

If you have not selected the [Save Server Description record option](#), the SAP Server Description dialog box appears. Type the name of the SAP server to which you want to connect. This value is the same as the **Description** field that displays in the [SAP Logon](#) configuration application. Press **Enter**. A log on dialog box appears.



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



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

SAP conversion options (Versions 6.x)

Save Password: Select to save the encrypted password. If this check box is not selected, you are prompted for a password during conversion.

VB Script: Select to generate Visual Basic Script for debugging outside QALoad . If this option is not selected, you receive C++ scripts that can be used for playback within QALoad .

Build SAP Libraries: Click the Build button to generate the QALoad SAP libraries based on your version of SAP. If you receive [linking errors while validating or compiling](#), you should click this button.

SAP 6.x command reference

QALoad provides descriptions and examples of the various commands available for an SAP script. For details, refer to the Language Reference Help section for [SAP 6.x](#).

Advanced scripting techniques for SAP

Required commands

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


```
SET_ABORT_FUNCTION(abort function);
DEFINE_TRANS_TYPE("capture.cpp");
HRESULT hr = CoInitialize(0);
if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info, "ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
```

Required commands for transaction restarting

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

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

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

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

Error handling and reporting

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

```
BEGIN_TRANSACTION();
try{
    SAPGuiConnect( s_info,"qacsapdb2");
    SAPGuiVerCheckStr("6204.119.32");

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

    DO_SLEEP(3);

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

    DO_SLEEP(6);

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

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

QALoad 5.02

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

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

...

DO_SLEEP(10);

SAPGuiPropIdStr("wnd[0]/usr/cntlIMAGE_CONTAINER/shellcont/shell/shellcont[0]/shell");
SAPGuiCmd1(GuiCtrlTree,ExpandNode,"0000000003");
SAPGuiCmd1(GuiCtrlTree,PutSelectedNode,"0000000004");
SAPGuiCmd1(GuiCtrlTree,PutTopNode,"Favo");
SAPGuiCmd1(GuiCtrlTree,DoubleClickNode,"0000000004");
SAPGuiCheckScreen("SESSION_MANAGER","SAPLSMTR_NAVIGATION","SAP Easy Access");
SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
SAPGuiCmd0(GuiButton,Press);
SAPGuiCheckScreen("SESSION_MANAGER","SAPLSPO1","Log Off");
} // end try
catch (_com_error e){
    char buffer[1024];
    sprintf (buffer," EXCEPTION 0x%x %s for VU(%i)\n",e.Error(),
            (char *)e.Description(), S_task_id);
    RR__FailedMsg(s_info,buffer);
} // end catch
END_TRANSACTION();
```

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

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

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

SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
BEGIN_TRANSACTION();

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

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

END_TRANSACTION();
```

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

```
SET_ABORT_FUNCTION(abort_function);
DEFINE_TRANS_TYPE("capture.cpp");
HRESULT hr = CoInitialize(0);
if( hr != ERROR_SUCCESS )
    RR__FailedMsg(s_info,"ERROR initializing COM");
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
SAPGuiConnect( s_info,"qacsapdb2");
SAPGuiPropIdStr("wnd[0]/usr/txtRSYST-BNAME");
SAPGuiCmdl(GuiTextField,PutText,"qaload1");
SAPGuiPropIdStr("wnd[0]/usr/pwdRSYST-BCODE");
SAPGuiCmdlPwd(GuiPasswordField,PutText,"~encr~1211616261");
SAPGuiCmd0(GuiPasswordField,SetFocus);
SAPGuiCmdl(GuiPasswordField,PutCaretPosition,3);
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmdl(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","SAPLSP01","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);
```

QALoad 5.02

```
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", "SAPLSP01", "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(%)\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 logons when the recording scenario differs from the run-time scenario. For example, if when you record, no users are logged on to the SAP environment and when you run the script, users are already logged on, the script may fail. To work around this issue, you can use the `SAPGuiPropIdStrExists` and `SAPGuiPropIdStrExistsEnd` commands to handle either scenario. This technique works by checking for the multiple logon dialog box from SAP and selecting the Continue option.

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

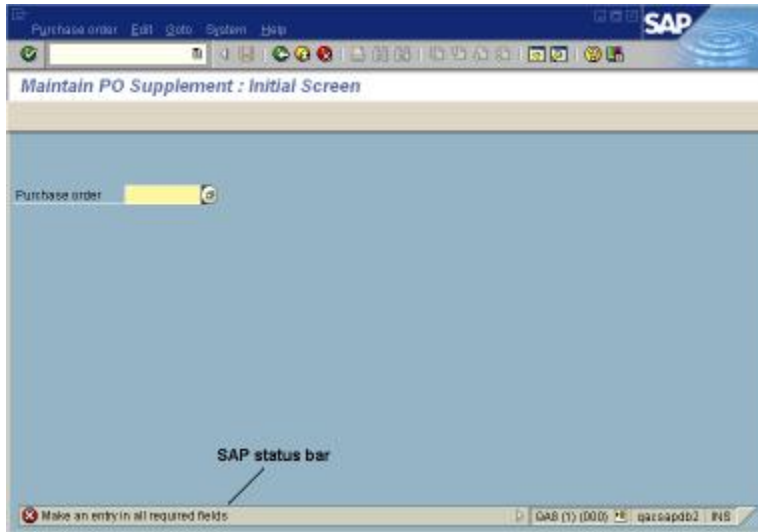
```
...
SAPGuiCheckScreen("S000","SAPMSYST","SAP");
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");

    DO_SLEEP(24);

    SAPGuiCmd0(GuiRadioButton,Select);
    SAPGuiCmd0(GuiRadioButton,SetFocus);
    SAPGuiPropIdStr("wnd[1]/tbar[0]/btn[0]");
    SAPGuiCmd0(GuiButton,Press);
    SAPGuiCheckScreen("S000","SAPMSYST","License Information for Multiple Logon");
SAPGuiPropIdStrExistsEnd("wnd[1]/usr/radMULTI_LOGON_OPT2");
...
```

Checking the SAP status bar

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



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

The `SAPGuiCheckStatusbar` command is used in the following script example:

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

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

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

if (bRetSts)
    RR__printf(" True\n");
else
    RR__printf(" False\n");
...

```

Object life span


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

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


Tuxedo

Tuxedo recording options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

 **Note:** If you choose this option and the application under test generates traffic before the first Windows screen displays, you must also select the **Capture Initialization Phase** check box on the [Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box](#).

Automatic: Select this option for QALoad to automatically start your application for recording, allowing you to record early application startup activity. This is the recommended method of recording, because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. Select this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

 **Note:** If you are recording from a PeopleSoft application, you must use the Automatic Program Startup method to successfully record any PeopleSoft sub-windows that make Tuxedo calls.

Command Line: If you chose Automatic Program Startup, enter the command line of your Tuxedo application. You can also use the browse button to locate your application.

If you are capturing from a PeopleSoft application, ensure that all of the parameters used with the application are appended to the command line.

Working Directory: Enter the working directory of your Tuxedo application or the directory of any additional files your application may require that do not reside in the application's path environment.

Tuxedo Version: Specify the Tuxedo version that is used for automatic and user-started program startup.

[Tuxedo conversion options](#)

Consolidate Checkpoints: Select this option to consolidate checkpoints. If you do not select this option, calls to the same service name generate more than one checkpoint.

Omit Asynchronous Calls (tpacall): Select this option to omit the asynchronous Tuxedo calls (TPACALL) during conversion. If this option is not selected, all TPACALL's will be converted to TPCALL. A comment will be inserted in the script to denote where each TPACALL was converted.

Use PeopleSoft certificates: If your PeopleSoft application uses certificates, any certificate you record will not be valid for replay after you terminate the transaction (tpterm). Select this check box to automatically insert a valid replay certificate in the script in the place of each recorded PeopleSoft certificate.

Use system VIEW environment variables: Select this check box to use the current VIEW environment variables (VIEWDIR, VIEWFILES) rather than the values used in the capture file. This option is useful when converting on a system other than the one which performed the original recording.

Tuxedo Version: Select the version of Tuxedo to use when running your script.

Convert FML Field Identifiers into Field Names: Select this option to identify the FML field name associated with each FML field identifier in the script during conversion. The FML field name is inserted into the script as a comment immediately following the associated FML field identifier.

Field Table Names: Enter or browse for the field table names used in your Tuxedo application. If the field table names are not given on the command line, then the program uses the FIELDTBLS environment variable as the list of field tables to be converted and the FLDTBLDIR environment variable as the list of directories to search for the files. FIELDTBLS specifies a comma-separated list of field table file names.

WSTUX32.DLL: Enter or browse for the path to the WSTUX32.DLL file if it does not reside in a directory within the application path. The WSTUX32.DLL is required to convert FML field identifiers to field names.

Field Tables: Enter or browse for the path of the field table name specified in the Field Table Names field.

[Setting up QALoad to run Tuxedo scripts on UNIX](#)

After installing the QALoad UNIX Player and utilities, you should ensure that the following environment variables are set prior to starting the Player Agent (loadagent):

Platform	Environment Variable	Value
All Platforms:	WSNADDR	<srvrname-or-IPaddr>:<port#>
	TUXDIR	<path>/<tuxedo dir>
AIX:	LIBPATH	<playerdir>/lib:<TUXDIR>/lib
HP-UX:	SHLIB_PATH	<playerdir>/lib:<TUXDIR>/lib
Linux:	LD_LIBRARY_PATH	<playerdir>/lib:<TUXDIR>/lib
Solaris:	LD_LIBRARY_PATH	<playerdir>/lib:<TUXDIR>/lib

Setting environment variables on UNIX systems depends on your login shell. For example:

```
! For ksh: export SYBASE_ROOT=/dir/tuxedo
! For csh: setenv SYBASE_ROOT /dir/tuxedo
```

The TUXDIR environment variable points to the directory where the Tuxedo workstation software has been installed. The WSNADDR environment variable specifies the network address(es) of the workstation listener process by which the client gains access to the application. For each UNIX platform, update the appropriate library path variable to include the library directory for the particular version of Tuxedo.

Scripts will be automatically downloaded to the Player machines by the Conductor and compiled, if necessary, at test execution time.

During the automatic script download and compile, if a script compile error occurs, there will be a scriptname.err file generated in the scripts directory.

To compile a script by hand, use the Rmake command. The syntax is as follows:

```
Rmake <scriptdir>/<scriptname>
```

or

```
Rmake <scriptdir>/<scriptname>
```

[Replaying a script with PeopleSoft certificates](#)

Because a PeopleSoft certificate is only valid until the transaction is terminated, a recorded certificate will not be valid at replay time and a script containing a recorded certificate will fail.

To keep a script from failing due to an invalid certificate, QALoad must automatically substitute the recorded PeopleSoft certificate with a valid replay certificate in each Tuxedo request that contains the certificate.

To automate the process of substituting the recorded certificate with a valid replay certificate, select the option Use PeopleSoft certificates on the Tuxedo tab of the Convert Options Wizard before attempting to convert your capture file to a script.

For more information, see the topic [Setting conversion options](#)

[Tuxedo command reference](#)

QALoad provides descriptions and examples of the various commands available for a Tuxedo script. For details, refer to the Language Reference Help section for [Tuxedo](#).

[Advanced scripting techniques for Tuxedo](#)

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. QALoad 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, QALoad Tuxedo scripts automatically handle buffer management. Instead of having to work with buffer pointers, QALoad'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 `Do_Tuxtpcall` command.

```
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. QALoad 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, QALoad checks to see whether the buffer associated with a specified buffer index was previously allocated. If QALoad 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 QALoad 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 retrieved 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`.


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 QALoad'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;

```

 **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.

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 QALoad's Convert facility places in the script or for data you place in the script.

The following QALoad 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~

Uniface

Uniface recording options

Uniface executable: Enter the full path or browse to the Uniface 7 executable that is used by the application you want to record. For example: `c:\usys72\bin\UNIFACE.exe`.

Working directory: Enter or browse to the working directory of your Uniface application or the directory of any additional files your application may require that do not reside in the application's path environment.

Initialization (.ini) file: Enter or browse to the full path to the Uniface application's initialization file.

Assignment (.asn) file: Enter or browse to the full path to the application's assignment file. For example: `c:\usys72\project\myapp.asn`.

Command line statement: Type command line options that should be used at application startup, including the command that is used to start the application. For example: `warehouse 1 1 control use=control dnp=tcp:`

Uniface conversion options

Includes: Type the full path or browse to the directory that contains the database include files.

Libraries: Type the full path or browse to the directory that contains the database library files.

Generate Uniface lists: Uniface can handle internal list structures. Select this check box to convert strings containing list items to a succession of DO_URB_xxx calls that manipulate Uniface lists.

Show output parameters: Select this option for the converted script to contain the output parameters of an operation as commented lines.

Insert trace messages as comments: Select this option for the converted script to contain the recorded content of the message frame as commented lines.

Uniface command reference

QALoad provides descriptions and examples of the various commands available for a Uniface script. For details, refer to the Language Reference Help section for [Uniface](#).

Winsock

How QALoad handles DO_WSK_Send commands

QALoad displays the contents of a DO_WSK_Send command as a string in a Winsock script. Some of these strings are very large, which can cause a compiler error (fatal error C1076: compiler limit: internal heap limit reached) if there are several large strings in a single script.

To avoid this compilation error, QALoad does not allow strings that are displayed in a Winsock script to be more than 12,000 characters. If a DO_WSK_Send command has a send buffer larger than 12,000 characters, its buffer is broken into smaller strings during the conversion. These smaller strings are then copied into a char buffer named "SendBuffer", which is sent in the DO_WSK_Send command. The size of the SendBuffer variable, by default, is declared as the size of the largest DO_WSK_Send + 1000. For example:

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
    /* Declare Variables */
    char SendBuffer[22139]; //Largest send is 21139 + 1000

    ...

    ...

    strcpy(SendBuffer, "$$ ..."); //Assume a large string, shortened for this example
    strcat(SendBuffer, "$$ ...");

    /* 12675 bytes */
    DO_WSK_Send(S1, SendBuffer);

    ...

    ...

    strcpy(SendBuffer, "$$ ...."); //SendBuffer is reused
    strcat(SendBuffer, "$$ ....");
    strcat(SendBuffer, "$$ ....");

    /* 21139 bytes */
    DO_WSK_Send(S1, SendBuffer);

    ...

    ...

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

Handling Winsock application data flow

Frequently, server programs return unique values (i.e., SessionID) that vary with each execution of the script and may be vital to the success of subsequent transactions. The following scripts demonstrate how this can be done.

Original Script

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);
#ifdef NULL
#define NULL 0
#endif
int rrobot_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 Script

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);
#ifdef 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);

```

QALoad 5.02

```
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[nBytesReceived] = '\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(PPLAYER_INFO *s_info)
{
RR_printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
}
```

Winsock recording options

User Started: Select this option if you would like to start your application manually for recording, either before or after you start recording. Because this method may fail to record your application's initial calls, Compuware recommends you use the Automatic option instead. Select the User Started option when you do not know the full application startup name and command option parameters or when the application spawns off processes that generate traffic that you want recorded.

Notes:

If you run a character-based application in a DOS window, the Script Development Workbench does not record the API calls.

If you choose this option and the application under test generates traffic before the first Windows screen

displays, you must also select the **Capture Initialization Phase** check box on the [Workbench Configuration tab of the Configure QALoad Script Development Workbench dialog box](#).

Automatic: Select this option if you want QALoad to automatically start your Winsock-based client, allowing you to record early application startup activity. This is the recommended method of recording because it takes advantage of QALoad's enhanced abilities to handle various multi-threaded programming techniques. When you select this option, the QALoad Script Development Workbench records the API calls that occur before the client enters its message loop. Select this option to record traffic from just one application. This option limits the recording output to just the traffic generated by the application, not including the traffic that is generated by processes spawned by the application.

Command Line: Enter the command line of your Winsock-based client. Note that if you enter the full path, QALoad automatically enters the path in the Working Directory field.

Working Directory: Enter the working directory of your Winsock-based client, if necessary.

Capture: Select the Winsock version to record.

Set IP Addresses: Click this button to open the [Add/Delete IP Addresses dialog box](#), which you can use to specify the IP addresses and ports on which you want to record Winsock API calls or that you wish to exclude from recording.

[Winsock conversion options](#)

There are no specialized conversion options for Winsock.

[Winsock command reference](#)

QALoad provides descriptions and examples of the various commands available for a Winsock script. For details, refer to the Language Reference Help section for [Winsock](#).

[Advanced scripting techniques for Winsock](#)

Understanding data representation in the script

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

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

1. The character is compared to the "space" character in the ASCII table, which has a decimal value of 32. If the character's value is less than 32, the following steps are taken:
 - b. If the character is "\r", "\n", "\t", or "\f", it is represented in the script as a normal C escape character.
 - c. If the character is either "\^" or "\^", it is represented in the script as an octal character. For example, the values would be "\034" and "\036", respectively.
 - d. If the character's value is less than 32 and it does not meet the descriptions in a) and b) above, it is represented in the script as a control character. For example, if the character is a null character, it is represented in the script as "\^@".
2. If the character's decimal value is between 32 (the "space" character) and 126 (~), it displays in the script as a standard readable ASCII character, with the following exceptions:
 - If the character is "\", which has a decimal value of 92, it is represented as "\\ " in the script.
 - If the character is "\"", which has a decimal value of 34, it is represented as "\" " in the script.
 - If the character is "\^", which has a decimal value of 94, it is represented as "\^" in the script.
3. If the character has a decimal value of 127, which corresponds to Delete (DEL), it is represented as "\^" in the script.

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

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

^[033	27	ESC
\034	034	28	FS
^]	035	29	GS
^_	037	31	US
	040	32	SP
\"	042	34	"
\\	134	92	\
^^	136	94	^
^?	177	127	DEL

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

Handling Winsock application data flow

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

Original code

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

```

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

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

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

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

#ifdef NULL
#define NULL 0
#endif

```

QALoad 5.02

```
int rrobot_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);

#ifdef NULL
#define NULL 0
#endif

int rrobot_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.
////////////////////////////////////

```

QALoad 5.02

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

/* 34 bytes */
//DO_WSK_Send(S1, "SessionID=jrt90847:^B^@^@^B^@^@^A^@^@^");
/* 15 bytes: ID Accepted#\r\n */

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

END_TRANSACTION();

REPORT(SUCCESS);

EXIT();

return(0);
}

void abort_function(PLAYER_INFO *s_info)
{
RR_printf("Virtual User %i:ABORTED.", S_task_id);
EXIT();
}
```

Modifying QALoad's functions to incorporate dynamic data

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

Scenario 1:

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

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

Example

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

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

...

BEGIN_TRANSACTION();
GET_DATA();

...

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

strcpy(temp, VARDATA(1));

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

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

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

QALoad 5.02

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

...

BEGIN_TRANSACTION();

...

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

/* 21 bytes: You are now connected */

DO_WSK_Expect(S1, "d");

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

temp = Response();

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

x = ResponseLength();

/* The line below will print the length of the response and the actual response */
RR_printf("length = %d, and response= %s",x, temp);
DO_WSK_Closesocket(S1);
```

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

Using the DO_WSK_Recv() command

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

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

Example

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

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

...

BEGIN_TRANSACTION();

...

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

/* 21 bytes: You are now connected */

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

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

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

Parsing server replies for values

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

Scenario 1:

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

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

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

```

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

...

BEGIN_TRANSACTION();

...

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

/* 23 bytes: sessionid=1234567890abc */

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

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

Scenario 2:

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

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

Example

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

```
/* Declare Variables */
char temp[15];
...
BEGIN_TRANSACTION();
...
DO_WSK_Socket(S1, AF_INET, SOCK_STREAM, IPPROTO_IP);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_OOBINLINE, 1);
DO_WSK_Connect(S1, "127.0.0.1", 90, AF_INET);

/* 16 bytes: xxx0123456789yyy */
memset(temp, '\0', 15);
DO_WSK_Expect(S1, "yyy");
ScanSkip(3);
ScanString(10, temp);
RR_printf("string=%s", temp);
DO_WSK_Closesocket(S1);
```

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

WWW

Visual Navigator

Visual scripting concepts

Introducing visual scripting

Visual Navigator for WWW is QALoad's easy-to-use visual interface to QALoad's powerful script development tools. Visual Navigator for WWW renders your recorded C-based transaction in a tri-paned, browser-like environment similar to popular visually-oriented development tools, with icons representing all the elements of your script. In fact, you could set up and run a WWW script without ever having to modify a C-based script.

With Visual Navigator's advanced editing features, you don't have to know the syntax of QALoad's command set or your HTML requests or responses to customize your script. You can quickly and easily:


- ! See what URL calls were made and what type they were (for example a POST or GET statement)
- ! See what information was passed in a call
- ! See what replies/pages were returned
- ! Add checkpoints or comments into your script
- ! Move the begin/end transaction statement
- ! Move the synchronize statement
- ! Edit an HTTP header
- ! Set particular flags and commands
- ! Add datapools
- ! Variablize your script
- ! Extract information from a reply to use in subsequent calls
- ! Save your script and go back to it at any time for further editing

! Create a C-based script file, if you like

[Show me the Visual Navigator](#)

Looking at a transaction loop

The transaction loop is the portion of your script that is played back repeatedly, representing multiple users making requests. The elements in your transaction loop depend on what was originally recorded on each page you requested. You can move the transaction loop up or down in the tree-view using the arrow buttons, to allow certain requests to be moved in or out of the Transaction Setup area, where they will be executed before beginning the transaction loop.

 **Note:** The following graphic does not show all the possible script elements, but gives a good representation of what your transaction loop might look like in the Visual Navigator.

High-level script items

There are three high-level script items in the transaction loop that represent the web pages you've recorded. NavigateTo, HTML Pages, and XML Requests:

NavigateTo: This is always the first item under the Transaction Loop element, and is always denoted with an arrow icon. It lists the URL that was typed into the web browser at the start of recording. This specifies the first request to be made. The result of this request is the next item in the tree, which is generally an HTML Page item.

If the first item is an HTTP request for XML data, it will appear as an XML Request item in the tree.

Page (HTML): Following NavigateTo there will typically be a set of HTML Page items, which are always denoted with a globe icon underneath the Transaction Loop element. These represent pages visited while the transaction was being recorded.

The form-view (bottom pane) lists the request's reply status, the requested URI, and the associated checkpoint name for the page returned.

HTML Page items can be parent to a number of script items in the tree-view, such as Action items. For more information about sub-items that can exist under a Page item, see [HTML Page sub-items](#).

XML Request: Requests for XML documents are denoted by a document/arrow icon underneath the Transaction Loop element. These represent the requests for XML data made during the transaction that was recorded. XML Request items can be parent to a number of lower-level script sub-items in the tree-view, such as Header and Cookie items and the XmlReply document item. See [XML requests](#) to learn about sub-items that can exist under an XML Request item.

What is variablization?

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

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

- ! If an ID string is returned from the server and that ID is then used as part of future requests to the server, and each virtual user may get back a different ID from the server, you could use a variable to use a specific ID. You could extract an ID from the reply, place it into a variable, and then use the value in that variable in place of the actual ID for future requests, ensuring you only use the ID you specify.

There are likely a number of values in your script that can be variablized. Those values are noted in the tree-view (bottom pane) with var....

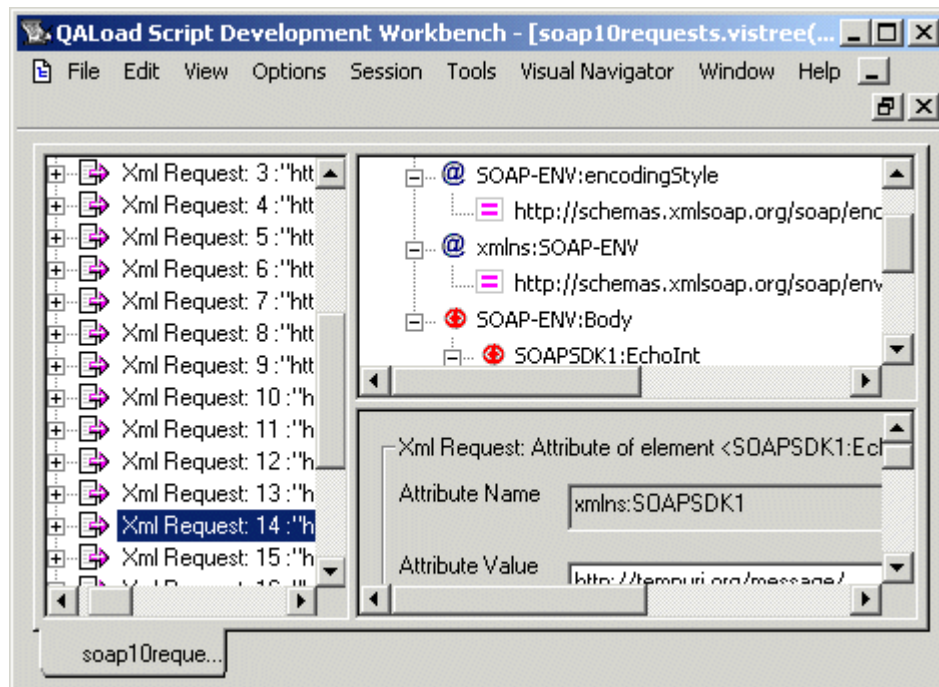
For details on how to create and maintain datapool files and variables, see [Datapools and variables](#).

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request, or an HTTP POST request with an XML document as the post data, then the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

- **Note:** To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the **View** menu.



Streaming media in Visual Navigator


If you selected the Streaming Media option on the WWW Advanced conversion options dialog box before recording your script, and the recorded transaction contains RealOne Player or Windows Media streaming requests, your streaming media request will be presented as a Page in the tree-view, similar to the following graphic:

The form-view (bottom pane) for a streaming media page will show the title Real Media Request or Windows Media Request to indicate the type of request you recorded, and will list the following fields:

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

Play Media Request: Select this check box for the virtual user to process the RM or ASX file that is received and make the necessary requests to duplicate what the client performed while receiving the streaming media. If this checkbox is not selected, then no further processing is performed after receiving the RM or ASX file.

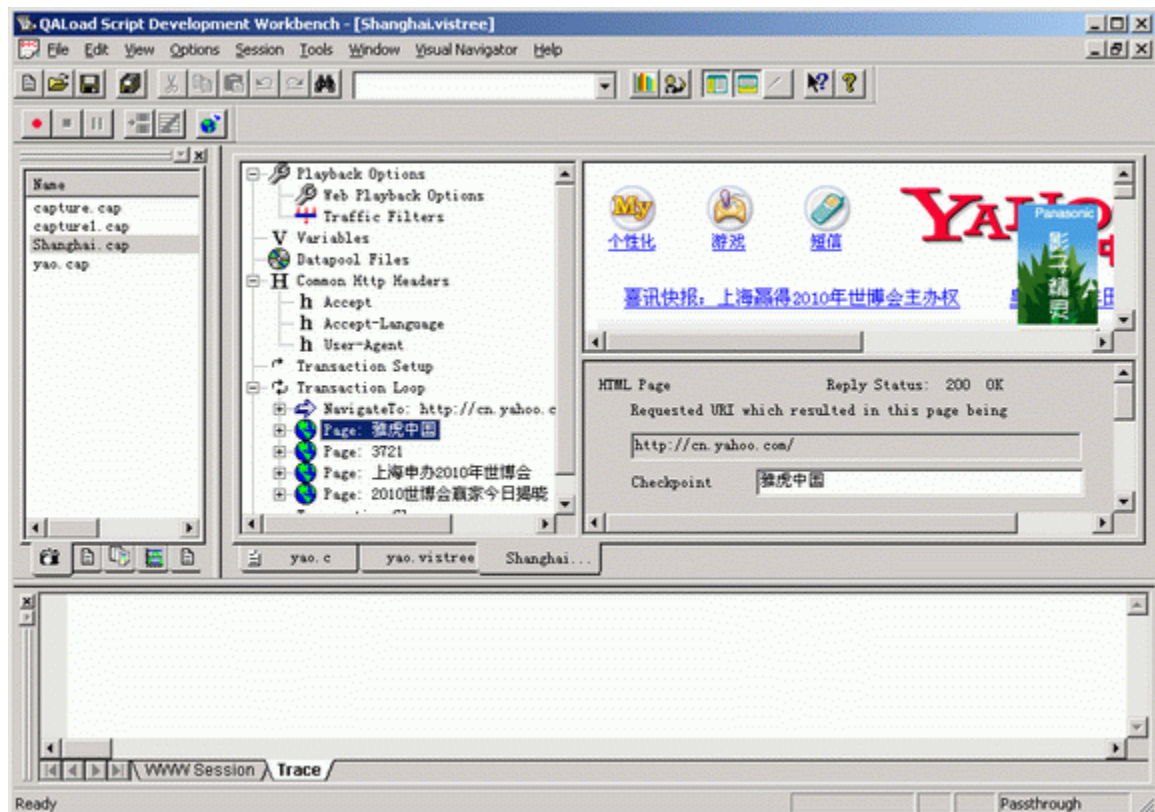
Play Requested Media for N seconds: You can specify how much of the streaming media file the virtual user should play, in seconds, before moving on to the next request. A value of zero indicates that the entire media stream should be played.

 **Note:** While a virtual user is playing a media request it will not make any other requests in the transaction loop. This may be different than what the user performed when recording the transaction because a browser is capable of spawning the streaming media player as a separate executable which can execute at the same time that the user continues to make further web requests in the browser.

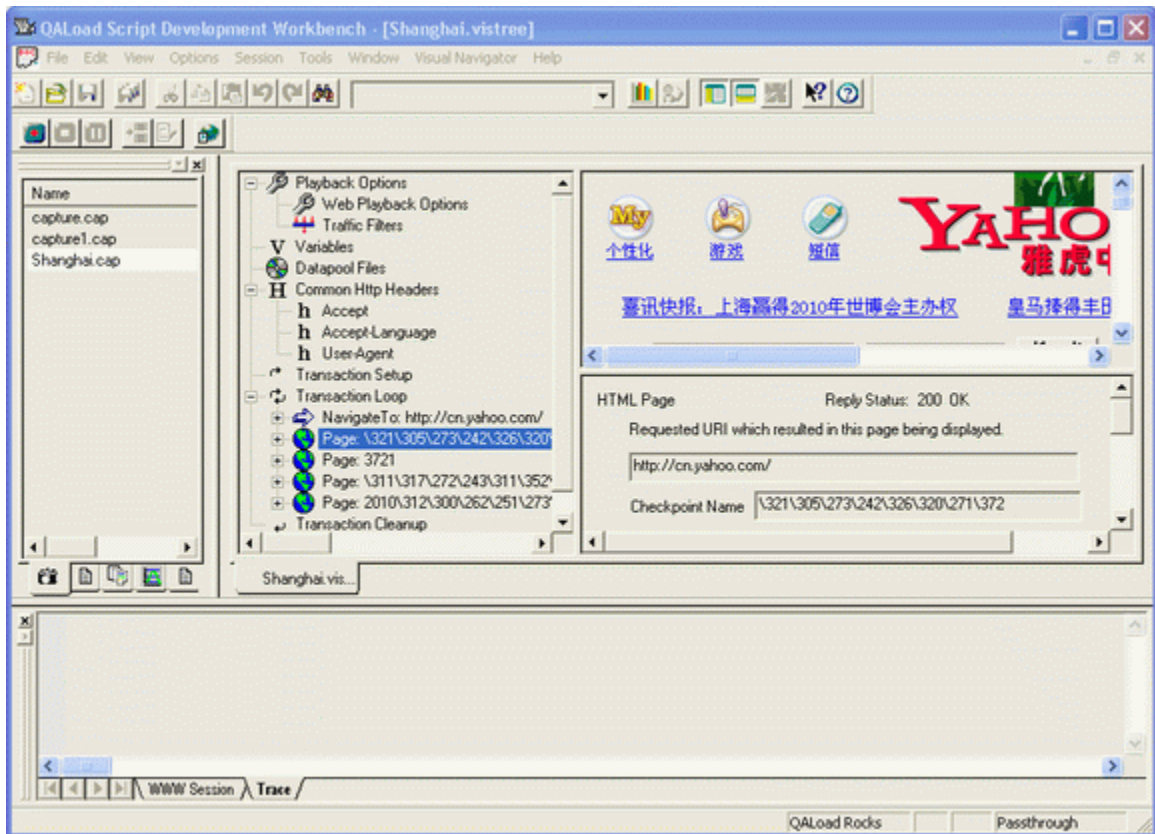
DBCS and Visual Navigator

The Visual Navigator handles both native and encoded characters. (See [DBCS Support in QALoad](#) for more information about DBCS support.)

The following graphic shows how the Visual Navigator provides native character support. In the following graphic, both English and Chinese characters are displayed in the Workbook Pane.



The same capture file, Shanghai.cap, is open in the graphic below. Here, the Visual Navigator displays the Chinese characters in encoded format.



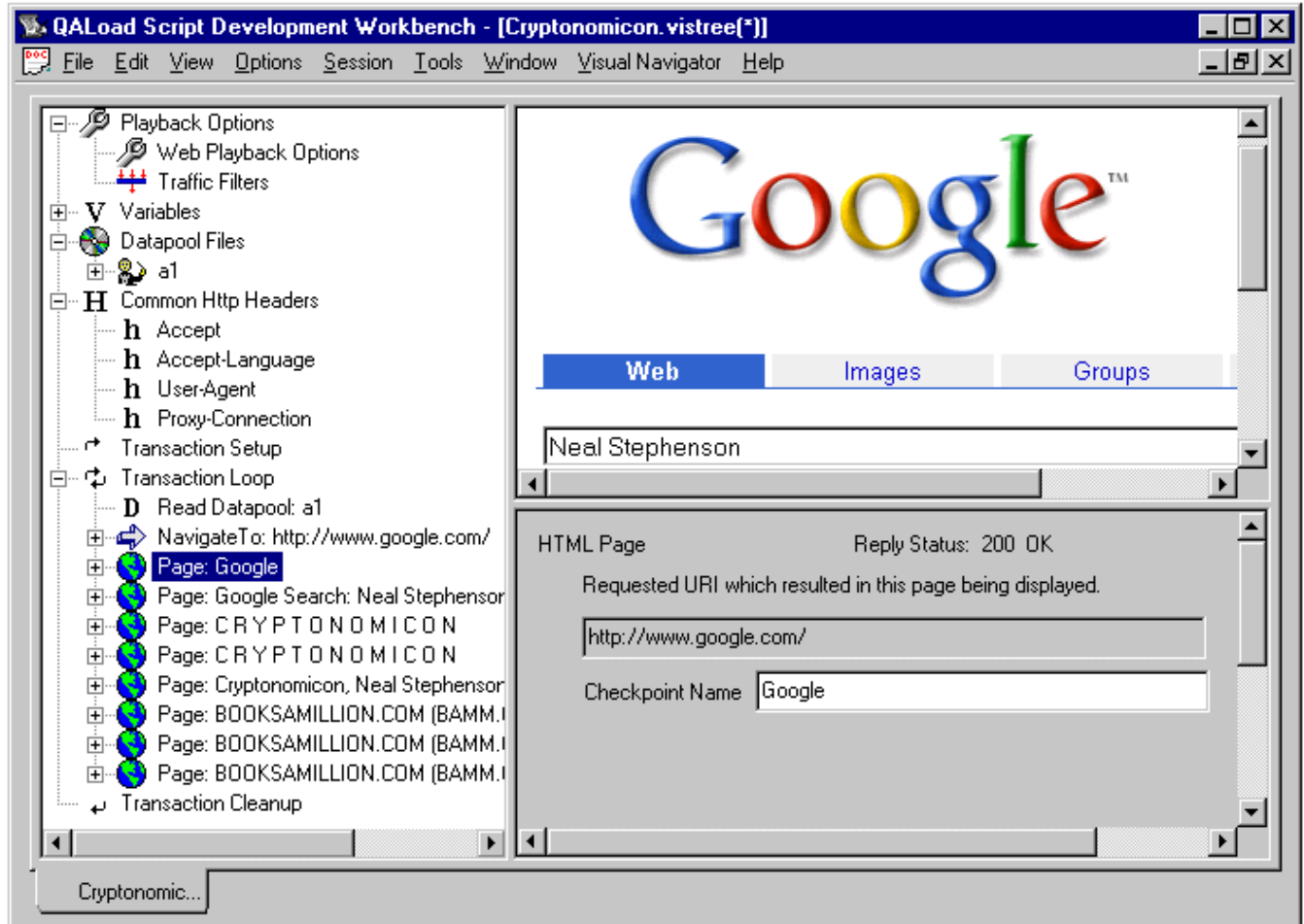
Visual Navigator's interface

The Visual Navigator

The visual scripting interface, called the Visual Navigator, has three panes that represent different aspects of your script, and menu items that offer you additional functionality. Using the Visual Navigator to develop your scripts makes your job easier. For example, searching through lines of code to locate a particular button you clicked on a particular page can take a long time, but using the Visual Navigator you can simply click through the pages to locate that button. In fact, you can develop your whole script – recording, variablizing, converting, compiling, and running it – all from the Visual Navigator's interface without ever writing a line of code.

For a brief explanation of each Visual Navigator pane, click on the panes in the graphic below. For more information about a pane, use the links listed after the graphic.

Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the **View** menu.



Visual Navigator's menus

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

Visual Navigator menu

The Visual Navigator menu is the main menu for Visual Scripting. Access the Visual Navigator menu from the Script Development Workbench's main menu, or by right-clicking on any item in the Visual Navigator tree-view (left pane).



Datapools and Variables: Opens the Datapools and Variables dialog box, where you can add, delete, or modify datapool files or variables.

View Script File: Opens a window showing the C++ (.cpp) file based on what is currently showing in the Visual Navigator's tree-view. This is a read-only script.

Create Editable Script File: Creates an editable C++ (.cpp) script based on the current Visual Script. You can modify this script directly; however, it will not be updated if you then make changes to the script using the Visual Navigator.

Show Hidden Fields: Displays form fields that may normally have been hidden by the browser.

Show Redirected Pages (3xx): Will toggle whether or not redirected pages will be displayed. These are pages that come back with a reply status code of 3xx, for example: 302 Not Found.

Show Client Error Pages (4xx): Will toggle whether or not Client Error pages will be displayed. These are pages that come back with a reply status code of 4xx, for example: 407 Proxy Authorization Required.

Show Server Error Pages (5xx): Will toggle whether or not Server Error pages will be displayed. These are pages that come back with a reply status code of 5xx, for example: 503 Service Unavailable.

Insert Tree Item: Opens a sub-menu where you can choose to insert certain tree items into your script. For details, see [Inserting script items](#).

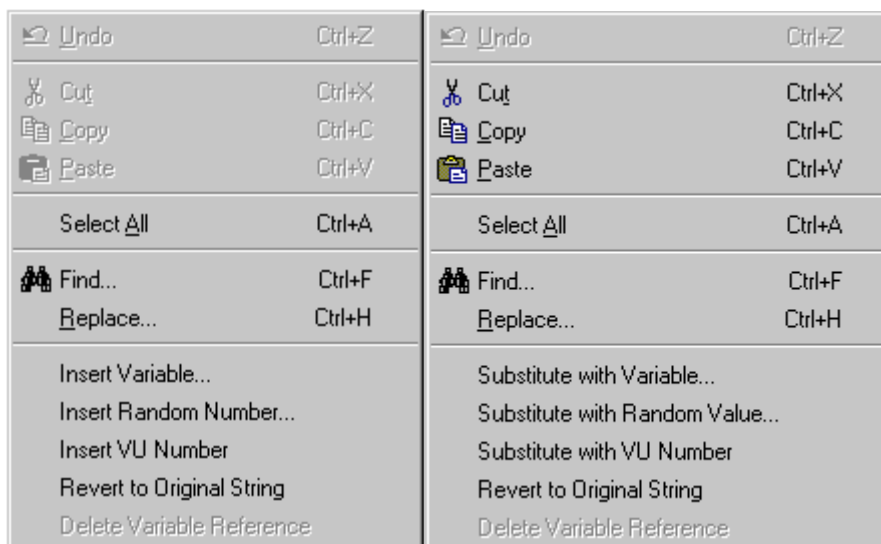
Delete Tree Item: Deletes the currently selected tree item. If the selected item may not be deleted from the script, this command is unavailable.

View Source: Opens a text window displaying the source code of the currently active HTML Page or Subrequest in the tree view.

Edit menu

The Script Development Workbench Edit menu provides special commands for Visual Navigator functionality as well as common Edit menu commands. Access the Edit menu from the main menu, or by right-clicking on an edit box that can be variablized in the Visual Navigator form-view (bottom pane). Fields that can be variablized are denoted with a Var button.

The commands on the Edit menu are dynamic and the availability of certain commands depends upon whether you have text selected and where your cursor is. The following graphics illustrate the difference:



Insert Variable/Substitute with Variable: Opens the Datapools and Variables dialog, allowing you to insert a variable or replace the selected text with a variable. Substituted text will refer to a local variable or datapool variable and will look similar to one of the following examples:

```
{Customer Number$}
{$Last Name:Customer Data$}
```

These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var next to it).

Insert Random Value/Substitute with Random Value: Opens the Random Number Tag dialog box where you can specify a range within which a random number should be generated for this value. The substituted text will look like this:

```
{$RANDOM:0:100$}
```

and it will produce a random number between the lower and upper limit each time it is executed.

These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var next to it).

Insert VU Number/Substitute with VU Number: Inserts, or replaces the highlighted text with, the following text:

```
{$VU_NUMBER$}
```

which will a virtual user number at runtime. Typically this will be combined with other text to form a larger string. For example:

```
Customer{$VU_NUMBER$}
```

will produce Customer1, Customer2, and so on, depending upon what the virtual user number is.

These commands are only available when the cursor is placed in an edit box on a tree-view item that can be variablized (you will see Var next to it).

Revert to Original String: Rolls the contents of the selected edit box back to when it was first created, usually when the recording was converted to a Visual Script.

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

Delete Variable Reference: Deletes the variable from the selected edit box. Note that you can also highlight a variable and press the Delete key to delete a variable within an edit box.

This menu item becomes enabled when you highlight a variable inside of an edit box.

Elements of a Visual Navigator script

Primary script elements

Elements of a Visual Navigator script

When you open a Visual Navigator script, you'll see standard elements of your script listed in the left pane. Each element can contain a number of script items, which in turn have attributes that are editable in some cases. This topic lists the major elements of a script, and links to additional help topics describing each element's associated script items:

The main elements of a Visual Navigator script are:

[Playback Options](#)

[Web Playback Options](#)

[Traffic Filters](#)

[Variables](#)

[Datapool Files](#)

[Common Http Headers](#)

[Common Content Checks](#)

[Transaction Setup](#)

[Transaction Loop](#)

[Transaction Cleanup](#)

Playback Options

Lists various settings related to playback. Settings are contained in the [Web Playback Options](#) and [Traffic Filters](#) items.

Web Playback Options

This item contains settings related to playback such as proxy settings, time out value, number of concurrent connections, baud rate emulation, and filters. This form is divided into three areas: Proxy Settings, Miscellaneous Playback Settings, and Memory Options.

Proxy Settings

This section of the form shows the proxy settings as they were originally recorded. This is the same information you could enter into the Internet Explorer or Netscape browsers to configure them.

Use Auto Configuration Script: If an auto-configuration script was used while recording, then this checkbox will be selected and the address of the auto configuration script will be shown. If an auto configuration script was not used, then the other proxy setting fields are enabled.

HTTP Proxy: The address of the proxy server machine and its port number.

SSL Proxy: The address of the SSL proxy server machine and its port number.

Exceptions: Separate entries with a comma, for example: company.sample.com, company2.company.com

Http Version: Specifies which protocol to use when making HTTP and SSL requests during playback. Options are Auto, 1.0, or 1.1. The version affects whether or not replies may come back chunked. Only HTTP 1.1 requests will receive chunked replies.

Username: A valid user ID to use if the proxy server requires authorization.

Password: A valid password corresponding to the user ID in the previous field, to use if the proxy server requires authorization. The password will be encrypted when recording.

Miscellaneous Playback Settings

This section contains various settings used for playback. The values on this form initially originate from the [WWW Convert Options dialog box](#); however, you can change some of the values on this form.

Server Response Timeout: The number of seconds to wait for a response from the server before considering it to be timed out.

Max Connection Retries: Specifies how many times a connection will be tried before it is considered failed.

Max Concurrent Connections: The maximum number of concurrent connections that can be opened to request documents. QALoad supports from 1 to 4 connections. This option allows QALoad to better simulate a browser's behavior while playing back a script.

Persistent Connection During Playback: When selected, QALoad will try to maintain an open connection to the server during playback. An open connection may better simulate the recorded transaction.

Caching: When selected, QALoad will simulate the browser's caching behavior.

Reuse SSL Session ID: When selected, QALoad will reuse the current session's communication

information (session ID) for all page requests within the transaction.

Use Transmission Baud Rate Emulation: Select to simulate a specific baud rate for transmission of requests, and then select a rate from the drop-down field. QALoad will then slow the transmission of requests appropriately in order to emulate the transaction rate of a modem.

Use Reception Baud Rate Emulation: Select to simulate a specific baud rate for reception of requests, and then select a rate from the drop-down field. QALoad will then slow the reception of requests appropriately in order to emulate the transaction rate of a modem.

Refresh Timeout (Seconds): A page will be considered a redirect if the META Refresh CONTENT field's value is less than the value in this field. This field can only be set in the [conversion options](#).

Memory Options

This section contains sections related to memory usage at the virtual user level.

Virtual User Memory Size: Indicates whether the current setting is for Typical or Minimize Memory Usage.

Memory Options: Click to open the [Memory Options dialog box](#), from which you can change from Minimize Memory Usage mode to Typical mode, or change Typical mode settings. Once a script is converted using Typical mode, the memory usage setting cannot be changed to Minimize Memory Usage.

Manually Select Subrequests: Indicates whether the current setting is Active or Not Active. This option can be modified on the Memory Options dialog box. When the option is Active, the following buttons are also available.


Select All Subs: Click to select all subrequests.

Clear All Subs: Click to clear all subrequests.

Traffic Filters

Traffic Filters allows you to filter out certain requests while playing back a script. For example, you may want to eliminate advertisement requests that are being made of Web servers other than the one you are testing.

The first time you convert a capture file, it will use the filters specified on the [WWW Advanced](#) conversion options dialog box and will eliminate any pages indicated as being filtered. Filter information entered on that dialog will be written to the Traffic Filters tree item. Any changes you make to this tree item will not affect items in the Visual Navigator tree but will be strictly for filtering subrequests at playback time.

 **Note:** Traffic filters do not affect which XML requests are written to the tree when a capture file is converted.

Two types of filters may be specified:

! Allowed Traffic

This list specifies a set of substrings for the type of URLs you will allow to be requested. The URL of every request made must include at least one of the specified substrings. The filter only applies to subrequests.

For example, if the list contained the strings "compuware" and "compuweb" then QALoad would allow the URLs "www.compuware.com\ info.htm" and "http://mycompuweb/weather.htm" to be requested, but not "www.google.com".

! Blocked Traffic

This list specifies a set of substrings for the type of URLs which should not be requested. If a URL to be requested contains any of the substrings in this list, then the request will not be sent. The filter only applies to subrequests.

For example, if this list contained the terms "doubleclick" and "popup_source" then QALoad would not allow the following URL to be requested: "http://www.doubleclick.com/myAdvertisement.htm".

The traffic filters form-view also lists the substrings that were converted to Additional Subrequests during conversion under the Traffic converted to Subrequests heading. This list is read-only and displays the settings from the [WWW Advanced](#) conversion options dialog box.

Variables

Lists local variables that have been created for this script. For more information about variablization, see [Variablizing a Visual Script](#).

Form View Fields

Variable Name: The name you provided to identify the variable.

Initial: The initial value for this field, before variablization takes place.

Datapool Files

Lists datapool files being called by the script. Each datapool listed has a list of variables under it representing columns in the datapool file. Datapools can be Local (specific to a single Player) or Central (available to all Players).

Common Http Headers

Lists headers that were recorded from at least 50% of your requests. These headers will be sent out with every request that is made at playback unless they are overwritten by a header of the same name underneath an individual request action.

You can insert new header items from the tree-view by clicking Visual Navigator>Insert Tree Item>Http Header. In addition, you can modify the values in the Http Header form in the right pane.

Common Content Checks

Lists common content checks, which apply to all replies sent by the server. Content checks enable you to verify whether the correct page was returned based on the existence or absence of a specific search string. You can also set content checks at the [page level](#). Click the Add New Content Check Item button in the form-view to add new common content checks.

Common content checks can include variables. Common content checks enable you to generate an error code on a set condition even if no individual page-level content checks are enabled. The search string is compared to the raw HTML returned by the server, so you may need to include HTML tags in your search to match the text that appears in the browser.

Transaction Setup

Lists any actions that occurred before the main transaction loop. Any items/actions that occur under this heading will be executed after the Synchronize but before the BEGIN_TRANSACTION(); statement at playback. For example, you may have logged in to a particular Web site and do not want to log in and out with every transaction at playback. You can move the Transaction Setup item in the tree-view by highlighting it and clicking the Move UP/Move DOWN buttons. The Transaction Setup can contain [client certificate tree items](#).

Client Certificate tree item

If the recorded transaction contains a line with a ssl-dientcert command, then Visual Navigator will create a Client Certificate tree item and place it directly beneath the Transaction Setup tree item.

The Client Certificate string can be modified or variablized in the form-view.

The Client Certificate item can also be moved up and down the tree like other tree items, such as checkpoints. This allows you to move it into the Transaction Loop area if you wish to change the certificate with each transaction.

A Client Certificate item will generate a script line similar to the following:

```
Set (EVERY_REQUEST, CERTIFICATE, "qaload_cl");
```

If the Requires Password check box is selected, the generated script line is similar to the following:

```
Set (EVERY_REQUEST, CERTIFICATE_PASSWORD, "~encr~250F7641455876");
```

Transaction Loop

Lists the requests in your transaction. All items/actions that occur under this heading will be placed between the BEGIN_TRANSACTION and END_TRANSACTION statements causing them to be repeated for as many times as the Conductor tells them to be. The transaction loop has a number of possible sub-elements, depending on the Web site you tested. For detailed descriptions of the elements that can be listed in a transaction loop, see [Looking at a transaction loop](#).

Transaction Cleanup

Lists actions that occur after the script has finished executing the appropriate number of transactions. Any items that occur under this heading will be placed after the END_TRANSACTION statement. For example, you may want to log out of a particular Web site after completing the appropriate number of transactions. You can move the Transaction Cleanup item in the tree-view by highlighting it and clicking the Move UP/Move DOWN buttons.

Transaction loop items

Synch

Inserts a Synch item immediately after the currently selected HTML Page. A Synch item represents a spot where all virtual users will pause during replay until all active virtual users have reached the same point. Once the virtual users are synchronized this way, the Conductor will instruct them all to continue.

A Synch item can be moved up or down the tree using Up/Down in the form-view.

IP Spoof

Inserts an IP Spoof item immediately after the currently selected HTML page.

In order for IP Spoofing to work with Visual Navigator, it is necessary to create or insert an existing local datapool file called QALOAD_IPSPOOF in the Visual Navigator tree-view. For more information about creating this datapool file and inserting it, see [Creating the QALOAD_IPSPOOF datapool file](#).

Read Datapool

Opens the Datapool and Variables dialog box, allowing you to choose which datapool to use, and then inserts a Read Datapool item immediately after the currently selected HTML Page.

You can move this item up or down the tree-view by clicking Up/Down in the form-view.

Form View Fields

The form-view (bottom pane) for a Read Datapool item displays the following information:

Datapool Name: The name of the datapool file that will be used. To use a different file, click **Select file...** and then choose or create another.

Datapool Variables: Lists the variables (fields) associated with this datapool file.

Checkpoint pair

Inserts a Begin Checkpoint item before the currently selected HTML Page and an End Checkpoint after the currently selected HTML Page.

Checkpoints are used to measure duration times for certain actions to be completed. You can move either the Begin or End checkpoint item to encompass several requests, if necessary. To move either item, highlight it and then click Up/Down in the form-view.

Form View Fields

The form-view fields for a Begin/End Checkpoint item lists the following information:

Name: The unique name of the checkpoint. When a checkpoint item is inserted, it is given a default name such as `UserCheckpoint 1`. This name can be changed. Changing the name of the Begin or End Checkpoint item automatically changes the name of the corresponding Begin or End item. In addition, deleting a Begin or End Checkpoint item will automatically delete the corresponding Begin or End item.

Show End/Begin Checkpoint: Click this button to quickly locate the Begin or End Checkpoint item that corresponds to the item currently selected.

Increment Variable/Decrement Variable/Reset Variable

Increments, decrements, or resets the value of a local variable.

Opens the Datapools and Variables dialog box, which allows you to select which variable to increment, decrement, or reset. It then inserts the appropriate item (Increment Variable, Decrement Variable, Reset Variable) after the currently selected HTML page.

Form View Fields

The form-view (bottom pane) for an Increment, Decrement, or Reset Variable item lists the following information:

Variable Name: The name of the variable being modified. To use a different variable, click `Select Var...` and then choose or create a different local variable.

Action: Describes the type of action to perform on the local variable. Options are:

- ! **Increment** — increments the value by 1.
- ! **Decrement** — decrements the value by 1.
- ! **Reset Value** — Type a value to replace the current variable with in the **New Value** field.

Debug Print

Inserts a Debug Print item after the currently selected HTML Page that will cause a string to be output to the Player window during playback. This can be useful for debugging a script while you are trying to variablize it so that it will replay correctly with multiple virtual users.

Form View Field

Debug Print Statement: If you simply type a string in this field, it will be printed in the Player's output window just as you typed it. If you insert a variable into the string, the value of the variable will be printed to the Player's window at playback so you can see its value. For example, to print the value of a datapool variable you might enter a string like the following:

```
Customer Name is {$First Name:Customer Data$} {$Last Name:Customer Data$}
```

which would print a statement like the following to the Player window at runtime: `Customer Name is Cindy Nelson.`

Comment

Inserts a Comment item after the currently selected HTML page. Type your comment into the form-view (bottom pane).

HTML Pages

HTML Page form-view

The form-view (bottom pane) for an HTML Page tree item contains the following information:

Reply Status: The code designating the status of the reply. For most pages that were returned correctly, this will be 200 OK.

Requested URI: This read-only field lists the URI which was requested that resulted in this page being displayed.

Checkpoint Name: If the page has a title, then it will be used as the checkpoint name. If not, the word Checkpoint will be used. To make sure all checkpoint names are unique, a number may be appended to the end of the checkpoint name.

Meta Refresh Required [] Seconds Before Redirection: If the Refresh Timeout option was selected on the [WWW Conversion Options](#) dialog box, this field displays the number of seconds that QALoad waits before it treats a META refresh request as a normal request. This field only appears when refresh timeouts are enabled.

HTML Page sub-items

The following script items can exist under a Page (HTML) item in the Visual Navigator's tree-view. Each possible page sub-item is listed below, along with descriptions for the fields that appear in the form-view in the right pane when you select the item in the tree-view.

In addition, a Page item can contain sub-items that you [insert manually](#) after recording the transaction.

[Content Check sub-item](#)

[PageCheck sub-item](#)

[AdditionalSubRequests sub-item](#)

[SubRequests sub-item](#)

[Cookies Set by Server sub-item](#)

[Sleep sub-item](#)

[Fill In Form sub-item](#)

[Action sub-items](#)

Content Check sub-item

Inserts a Content Check item for the currently-selected HTML page. Insert a content check to verify whether the correct page was returned based on the existence or absence of a particular search string in the server's reply for that page. Content checks can include variables. The search string is compared to the raw HTML returned by the server, so you may need to include HTML tags in your search to match the text that appears in the browser.

The top pane displays the source for the HTML page. You can easily select text in the top pane and add it to the content check definition by clicking the Copy from Source button.

Form-view fields

The form-view (bottom pane) includes the following fields:

Enable this content check: Select to enable the content check for the page. Content checks are disabled by default. After checking this box, choose whether the check succeeds if the string is contained or not contained in the page. Then type the string text in the box.

Copy from Source: Select text in the top pane and click this button to copy the selected text in the top pane to the text box.

Verify: Click to test whether the check succeeds for the text shown in the top pane.

PageCheck sub-item

Allows you to verify that the title of the page that was requested is correct.

Form View Fields

The form-view (bottom pane) includes the following fields:

Original Title: This read-only field displays the title of the recorded page.

Verify Page Title: Select this checkbox to enable/disable title verification at playback, then select the appropriate option:

Complete Title: The title of the page that is requested must match exactly with the original title.

Prefix: Matches the first characters of the title. Specify how many characters to match. At playback time QALoad will then compare the first N characters to the original title, allowing you to verify titles which might change slightly.

Suffix: Matches the last characters of the title. Specify how many characters to match. At playback time QALoad will then compare the last N characters to the original title, allowing you to verify titles which might change slightly.

AdditionalSubRequests sub-item

Some requests are contained in applets, ActiveX components, or other objects that are captured, but not played back by QALoad. These subrequests, which are not recognized as normal subrequests, are listed in the AdditionalSubRequests tree item.

Each additional subrequest item appears in the script as a pre-loaded subrequest just before the main action. As a result, the playback engine requests the main page, regular subrequests, and then the pre-loaded subrequests.

For example:

```
//----- REQUEST # 2 -----  
//  
// current page url is http://c96852d01/pda/  
//  
// Pre-load the following image requests before the next request is made.  
// These requests seem to have been made by javascript or applets associated  
// with the next page but will not be made automatically by the replay engine,  
// hence they are here in the script.  
//  
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,  
    "http://c96852d01/pda/images/LeftBackgrnd.jpg");  
  
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,  
    "http://c96852d01/pda/menuopen.gif");  
  
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,  
    "http://c96852d01/pda/menuclose.gif");  
  
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,  
    "http://c96852d01/pda/menuclose.gif");  
  
Set (NEXT_REQUEST_ONLY, ADDITIONAL_SUBREQUEST,  
    "http://c96852d01/pda/images/browsex.gif");
```

```
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://c96852d01/pda/images/browse.gif");
```

SubRequests sub-item

Lists all subrequests (such as images) that the page performed in order to be fully rendered in the browser. Subrequests cannot be changed and are shown strictly to provide detailed information about the requests that were made during the recording session.

Form View Fields

The form-view (bottom pane) for subrequests contains the following fields:

URI: Lists the URI of the subrequest that was made.

View Source: Click to open the source file associated with the reply of the subrequest in a text format. Image files will show up as mostly unreadable characters, but cascading stylesheet and JavaScript requests will be readable.

Cookies set by Server sub-item

If the reply from the server for the requested page contains a Set-Cookie command, it is listed here. This item cannot be modified, it is listed for your information only.

Form View Fields

The form-view (bottom pane) for cookies lists the following information:

Name: The name of the cookie.

Value: The value that the cookie should be set to.

Path: The path within the domain to which this cookie is assigned. This cookie will only be sent to the server if future request URIs have the same domain and this path. The path can be deeper but it must start with this path.

Domain: The domain to which this cookie is assigned. This cookie will only be sent to the server if future request URIs have this domain and the path specified above.

Expires: The cookie's expiration date, if it has one.

Sleep sub-item

Every page will have a Sleep item immediately before its Action item. The sleep value specifies how many seconds were spent viewing this page (or filling out a form) before an action was taken (such as clicking on a link or button).

Fill In Form sub-item

If a requested page contains a Form (html element) that was filled in by the user, then a Fill In Form item and its associated elements will be created in the tree-view. When this tree item is selected, a blinking frame will appear around the form in the browser-view (top pane).

Form View Fields

The form-view (bottom pane) for a Form element lists the following information:

Form Action String: The Action String associated with this form. For example, the URI which will be requested.

Form Number: The number assigned to the form, indicating which form will be used.

A Fill In Form item in the tree-view can include a number of sub elements. For details, see [Form sub-items](#).

Extract String sub-item

Insert an Extract String item when you need to extract information from a reply and store it in a variable to use in variablizing future requests or simply for logging the information. For example, if a string is located inside a JavaScript or in a hidden tag that is not visible in the browser, and it might change each time this page is requested, use an Extract String to extract the value.

When an Extract String item is inserted into the tree-view, the browser-view will display the HTML source for the page in which the item was inserted. Click Select New String, highlight the text to extract, and then click Done.

The string to extract will be recognized by the text preceding it and certain text following it. Anything in between will be extracted and saved into the local variable. When you select text initially, Visual Navigator will use 10 or more characters on either side of the extracted text to make the search string unique enough to find this copy of the extracted text. You can increase or decrease the size of these strings using the spin controls.

You must also specify a variable that will receive the extracted string at run time by clicking on Select Var.

Form View Fields

The form-view (bottom pane) for an Extract String item will list the following information:

Variable to receive string: The local variable that will receive the string extracted from the reply. The value will initially be None.

Select Var: Click to access the Datapools and Variables dialog box where you can select or create a local variable to use with the Extract String item.

Preceded by: The text preceding the string you want to extract.

Extracted: The string to be extracted on this particular reply.

Followed by: The text following the string to be extracted.

Select new string: Click to select a new string in the browser-view (top pane). After clicking this button, highlight the string text in the browser-view. The button text changes to Done. When you have selected the appropriate string, click Done. The Preceded, Extracted, and Followed by fields will be filled in automatically, and the string to be extracted will remain highlighted in the browser-view.

Frames

If an HTML page that is recorded contained frames, they will be represented in the tree-view (left pane) with a circle icon containing a capital F. A frame page will be indented beneath the page that is its parent. If you click on a frame icon in the tree-view, the corresponding frame will be highlighted in the browser-view (top pane) with a blinking frame around it for identification.

Duplicated frameset pages

Sometimes when a user clicks on a link or takes some other action inside of a frame, the new page that was requested simply replaces the contents of one of the frames already shown in the browser. To indicate that the frameset page (the main page that holds the frames) has not changed, Visual Navigator renames it Duplicated Frameset n. Where n is an identifying number for the frameset that is incremented as more frameset pages are duplicated.

Form view fields

The form-view (bottom pane) for a Frame Page item lists the following information:


Requested URI: The URI which was used to request the frame page

Frame Name: The name of the frame as indicated in the source HTML.

Action sub-items

Action sub-items

An Action item appears under each HTML Page except the last one in the script. This represents the action that the user took to get to the next page. Action items include:

- ! [NavigateTo](#) – Visual Navigator could not determine how the user accessed the next page (they may have typed a URL directly into the address bar, or a JavaScript may have caused the jump).
-  **Note:** A [NavigateTo](#) item is the first element that appears under the Transaction Loop element. This is because when the browser is launched during recording, the user must specify a starting address (typically by typing it into the Address bar).
- ! [Click On Link](#) – The user clicked on a text link
- ! [Click On Image](#) – The user clicked on an image link
- ! [Click On Button](#) – The user clicked on a Submit type button.
- ! [PostTo](#) – Data was sent to the server with a POST command but a matching submit button was not found. This may have been caused by a JavaScript.

Action items can contain various sub-elements. For details, see [Action item sub-elements](#).

NavigateTo sub-item

Specifies a URI to be requested. If the Script Development Workbench cannot determine how the next page was requested (typically due to a JavaScript making the request) then it will use a [NavigateTo](#) tree item instead of something more specific such as a [Click On Link](#).

Form View Field

The form-view (bottom pane) for a [NavigateTo](#) item lists the following information:

URI: The URI that was requested. This value can be variablized.

Use Link Encoding for CGI Parameters: If an HTTP GET request is the result of a direct request that includes the CGI parameters within the link, the parameters must be sent to the server in a different encoding called Link Encoding. In most cases, the Script Development Workbench automatically detects this type of request and selects this option. However, you can manually change this setting for troubleshooting. Generally, If a [Navigate_To](#) is associated with a form, use form encoding (do not select this check box); if a [Navigate_To](#) is associated with an anchor, use link encoding (select this check box).

Click On Link sub-item

If the user clicked on a text link or an image link, then a [Click On Link](#) action item will be inserted under that page. This is used to describe the action that was taken while on this page that resulted in the next page being requested.

When a [Click On Link](#) tree-view item is selected, the text or image in the browser-view (top pane) will be highlighted by a blinking frame to make it easier to locate. There are several types of [Click On Links](#):

- **Text Links** – One of the more common links in web pages are Text based links. These usually appear as underlined text.
- **Image Links** – An image can have a link, similar to text.
- **Client-Side Image Map** – A Client-Side Image Map is an image on a page that has multiple links associated with it. Each link is associated with a region, which can be any shape. When the user clicks on the image, the browser will determine which region was clicked on and will request the page linked to from that region.
- **Server-Side Image Map** – A Server-Side Image Map is an image on a page that has multiple links associated with it. Unlike Client-Side Image Maps, these links are stored on the server rather than the client. When a user clicks on the image, the browser will send the server the mouse coordinate relative to the top-left corner of the image. The server will then reply with the appropriate page.

Form View Fields

The form-view (bottom pane) for Click On Link items list the following information:

Link Type: Specifies the type of link found. For example, Text, Image, Client Side Image Map, or Server Side Image Map.

Link Text: Only available for text links. Shows the text associated with the link, as displayed in the browser-view (top pane). During playback, the script will search for a link using this text and then request the page specified by the link.

Image: Only available for image links. Shows the URL of the image that was clicked. When playing back the script, QALoad will search for an image with a link that originated from this URL and will request the page specified by the link.

If multiple matches to Target URI are found, use match: It could be possible to have multiple links to a single URI. Use the spin control to move to the next or previous link that matches the target URI. Selected links will be framed and the Link Text, or Image, or Image Region edit box (the edit box available depends on what type of link it is) will be updated.

Target URI: Displays the URI that will be requested when this link is clicked on.

X and Y Coordinates: Server Side Image Maps only. The mouse coordinates, relative to the upper left of the image, that were clicked on while recording.

Click On Button sub-item

Clicking on Submit is usually associated with entering values into a form (Fill In Form item). When the Click On Button tree-view item is selected, the associated button in the browser window (top pane) will be highlighted with a blinking frame, making it easier to locate.

Form View Fields

The form-view (bottom pane) for Click On Buttons (submits) lists the following information:

Identify by: Click the appropriate button to choose whether to identify the button by Name (internal name as indicated in the HTML), by Label (the text on the button), by Occurrence (e.g. the 3rd button), or by Image Source (in case this submit button is really an image). The edit Identify by buttons will change appropriately, depending upon which option you choose.

Prev Button/Next Button: If there are other submit buttons on the page, use Prev Button/Next Button to choose to use one of those other submit buttons. The name, label, or occurrence will be updated automatically.

Label: The Label field (or Name, Occurrence, or Image Source field) can be modified or variablized. If you variablize the field, clicking Next Button or Prev Button will not affect its contents. However if you choose to use one of the other options, different text will be displayed because variablization is specific to each method of identification.

If multiple matches to URL are found, use match: If QALoad finds multiple matches (for example, two buttons labeled Buy Me) then you can use this field to specify which match to use. For example you could specify that you want to click on the 4th button with a label of Buy Me when looking at a page with a lot of items for sale.

Original Label: The original label associated with the button that was clicked while recording.

Original Name: The original name (from the HTML source) for this button.

Form Action: The URL associated with a form. This is the address for the request that will sent when the submit button is clicked.

Submit Method: Whether this submit item was a GET or POST.

PostTo sub-item

If the recorded request was a POST request rather than a GET request and the Script Development Workbench could not find a matching Submit type button, then a PostTo tree action item will be inserted under the page. This can sometimes happen if the request is initiated by JavaScript.

Form View Fields

The form-view (bottom pane) for a PostTo item lists the following information:

URI: The URI which is being posted to.

Content Type: One of the following: DEFAULT, TEXT_PLAIN, or MULTIPART_FORM_DATA. This field is read-only.

Action item sub-items

Action item sub-elements

The following items can exist under Action items in the tree-view:

[Http Headers sub-item](#)

[Cookies sub-item](#)

[CGI Parameters sub-item](#)

[NTLM Authentication sub-item](#)

[Basic Authentication sub-item](#)

Http Headers sub-item

If a header exists under an Action item, then it will be sent for that request only. If the header has the same name as one of the common headers, then it will override the common header for this request only. It is possible to [insert additional HTTP headers](#).

Form View Fields

The form-view (bottom pane) for an HTTP Header lists the following information:

Name: The name of the HTTP header.

Value: The value of the HTTP header.

Cookies sub-item

When a Cookie item is a sub-element of an Action item, it contains a list of cookie items that were sent in the header of the request that the Action item made when recording. Cookies are added automatically by the browser based on the URI that is being requested. They are either set as a result of the previous reply (the server returned a Set-Cookie command), or they are set by JavaScript contained in the previous reply.

If the Cookie shown has a matching Set-Cookie item, then nothing will display in the script since the cookie is created automatically during playback. If there is no matching Set-Cookie item, then a Set-Cookie type statement will be generated in the script.

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

Form View Fields

The form-view (bottom pane) for a cookie item lists the following information. Both of these items are can be edited and variablized. You can also insert additional cookie items.

Name: The name of the cookie.

Value: The value of the cookie.

CGI Parameters sub-item

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

Form View Fields

The form-view (bottom pane) for CGI Parameter items lists the following information. Both these fields can be edited and variablized. You can also insert additional CGI Parameter items.

Name: The name of the CGI Parameter.

Value: The value of the CGI Parameter.

NTLM Authentication sub-item

Sometimes the pages being requested require NTLM Authentication, that is, the user will be presented with a dialog box asking for a UserID, Password, and Domain. This information is recorded and listed in the tree-view under the Action item that requires it.

Form View Fields

The form-view (bottom pane) for NTLM Authentication items lists the following information:

User: User name typed into the authentication dialog.

Password: Password typed into the authentication dialog.

Domain: Domain typed into the authentication dialog.

Basic Authentication sub-item

Sometimes the pages being requested require Basic Authentication, that is, the user will be presented with a dialog box asking for a UserID and Password. This information is recorded and presented in the tree-view under the Action item which requires it.

Form View Fields

The form-view (bottom pane) for Basic Authentication items lists the following information:

User: User name typed into the authentication dialog.

Password: Password typed into the authentication dialog.

Forms

Forms

Many pages that are used during a WWW load testing session will contain forms that a user must fill out and submit buttons that get clicked. QALoad will identify forms and the elements within them, as well as determine which submit button was clicked if there is more than one.

When a page contains a form that will be submitted, then a Fill In Form item will be inserted into that page's list of items in the tree-view (left pane). Underneath the Fill In Form item will be Form Element items such as Edit Boxes, Radio Buttons, and Check Boxes. Following the Fill In Form item will be either a Click On Button item or a PostTo item. For details about Form Elements, see [Form sub-items](#).

When a Fill In Form item is selected in the tree-view (left pane), Visual Navigator highlights the form with a blinking frame in the browser-view (top pane).

Form sub-items

A Fill In Form item in the tree-view can contain a number of sub-items, representing elements that can appear in forms on HTML pages.

The following sub-items can appear under a Fill In Form item:

[Hidden sub-item](#)

[Editbox sub-item](#)

[Selectbox sub-item](#)

[TextArea sub-item](#)

[Checkbox sub-item](#)

[Radio sub-item](#)

Hidden sub-item

Forms can contain hidden fields that do not show up on the page. These fields are not visible to the end user interacting with the browser, but they may need to be variablized for a load test, for example a field that contains a session ID may need to be variablized.

Form View Fields

The form-view (bottom pane) for a Hidden Field element lists the following information:

Name: The name of the hidden field

Value: The value of the hidden field.

Allow this hidden field to be variablized: Select to variablize this field. Click the var... button to select a variable.

Editbox sub-item

One of the more common elements in a form is an edit box. When this tree item is selected, QALoad will draw a blinking frame around the appropriate edit box in the browser-view (top pane). The edit box in the browser-view will show the value that was originally typed in when the transaction was recorded.

Form View Fields

The form-view (bottom pane) for an Edit Box element lists the following information:

Name: The name of the edit box.

Value: The value of the edit box. Any changes made to this field will be reflected in the edit box in the browser window.

Selectbox sub-item

A select box is often called a drop down selection box or list box. The form-view will appear slightly different depending upon whether the Select Box is capable of supporting multiple selections or not.

Form View Fields

The form-view (bottom pane) for a Select Box element lists the following information:

Name: The name (in the HTML) of the select box.

Items from the Select Box: Lists the items present in the Select Box in the browser-view. An item has a checkbox next to it to indicate if it has been selected. To change a selection, select or clear the checkbox. Your choices will be reflected in the browser. If the Select Box only supports one selection, then only the most recent selection is selected.

Variablized Selections: Edit boxes that allow the use of variables (local or from a datapool) to specify what options are chosen from the Select Box. For a multiple selection Select Box, it is possible to add up to six variables in addition to any item chosen using the check boxes.

For a single selection **Select Box**, a single edit box is provided to allow you to use a variable (local or from a datapool) to specify the option you want chosen from the **Select Box**.

TextArea sub-item

A Text Area item is a multi-line text box.

Form View Fields

The form-view (bottom pane) for a **TextArea** element lists the following information:

Name: The name of the Text Area field.

Value: The value of the Text Area field. Any changes you enter into this edit box will be reflected in the browser-view (top pane). To enter a linefeed, press **Ctrl+Enter**.

Checkbox sub-item

The form-view (bottom pane) for a **Checkbox** element lists the following information:

Name: The name of the Checkbox.

Value: The value of the Checkbox.

State: Reflects whether the box is checked (selected) or not. If the **State** is 1 (checked), then the **Name** and **Value** are passed along in the request to the server. If the **State** is 0 (not checked) then the **Name** and **Value** are not passed along. You can change the value of the **State** by clicking on the checkbox control in the browser-view (top pane).

Radio sub-item

The form-view (bottom pane) for a **Radio Button** element lists the following information:

Group Name: The **Group Name** is shared by all radio buttons that belong to the same group.

Value: The **Value** field is what differentiates one radio button from another. The group name and value of the selected radio button will be sent along with the request to the server. The **Value** of a radio button can be, and often is, different than the text shown in the browser.

use this value button: When you select a radio button in the browser-view (top pane) its value will display in this text box. Click this button to transfer this value into the above **Value** field.

XML requests

XML document-view

When you click on an **XML Request** item in the tree-view (left pane) the right pane becomes a document-view displaying a tree-view of details about the XML document requested or returned as the result of an XML request. Each individual XML item appears as a node in the XML document tree. XML elements can have child elements and these appear as child nodes of the XML element. Attributes of an element appear as child nodes of the element, with the attribute value appearing as a child of the attribute name.

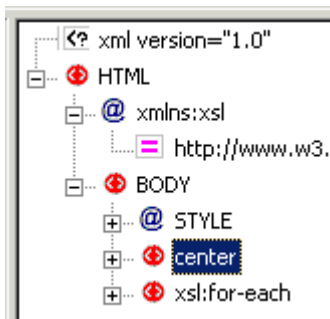
What if no XML data is associated with a request?

If there is no XML document associated with the XML request (for example, an HTTP GET) a message indicating that there is no XML to be displayed is shown in the XML document view.

How does the document-view relate to the form-view?

Selecting an item in the XML document tree will display the form-view details corresponding to that XML element type in the bottom pane.

Following is an example of XML request data displayed in a portion of the XML document-view:



XML form-view

When an XML request is displayed in the document-view (top pane) — as a result of an XML request item or XML reply child item being selected in the Visual Navigator tree-view — you can click on items in the document-view to display information about each in the form-view (bottom pane). If no XML item is selected in the document-view, the XML Page form-view will be displayed instead. For XML items, the form view display options depend on two things:

- ! what type of XML item is selected in the Visual Navigator tree-view (left pane): an XML request or an XML reply
- ! what type of XML item is subsequently selected in the XML document-view (top pane).

When an XML item is selected in the XML document-view, the value of that XML item is displayed in an edit box in the form-view. Some values — attribute values and text values — can be edited or variablized (that is, substituting one or more variables for the value in an XML request or selecting the return value from an XML reply item to be received by a variable for later use in the script). Text items, which are values between element tags, and attribute values represent volatile items in an XML document structure, used for passing values to and from Web Services, for example.

The following tables list the possible actions for XML items displayed in a form-view. Valid actions are determined by the XML item type and whether the item is from an HTTP POST request or from an HTTP reply.

In the following tables:

- ! If an item is *editable*, the value in the form-view can be changed and the new value will be used during replay.
- ! If a value can be *variablized*, a variable can be substituted for all or part of the value. The variable's value will be placed in the variable's location at replay. An example is a value received from an item from a previous XML document reply.
- ! If a variable can *receive* a replay value, the return value for the item can be placed into a selected variable during replay. The variable can then be substituted for an input value in a later XML request.

XML Request Items		
XML Request Item	Editable?	Can the Value be Variablized?
Declaration	No	No
DTD (Document Type Definition)	No	No
PI (Processing Instruction)	No	No
Comment	No	No
Element	No	No

Attribute (Name)	No	No
Attribute (Value)	Yes	Yes
Text	Yes	Yes

XML Reply Items	
XML Request Item	Can Variable Receive Replay Value
Declaration	No
DTD (Document Type Definition)	No
PI (Processing Instruction)	No
Comment	No
Element	No
Attribute (Name)	No
Attribute (Value)	Yes
Text	Yes

Using Visual Navigator

Creating a Visual Navigator script

Creating a Visual Navigator script

Using the Script Development Workbench, you can quickly and easily:

- ! Convert an old script to a Visual Script [How?](#)
- ! Record a new Visual Script [How?](#)

Converting previously-recorded scripts

You can convert existing C-based scripts to a Visual Navigator script.

To convert a previously-recorded script:

1. With a WWW session open, choose **Options>Convert**.
2. On the WWW tab, select the option **Enable Visual Scripting**.
3. Click **OK**.
4. On the Workspace Pane's Capture tab, right-click on the capture file to convert to a script. From the context menu, choose **Convert** (or **Convert As**).
5. You may be prompted to overwrite an existing script file. Click **Yes**.

The Script Development Workbench converts your capture file to a Visual Navigator script, opening it in the Workbook Pane.

Recording a Visual Navigator script

You record a Visual Navigator script the same way you record a regular QALoad script — by setting options to determine the behavior of QALoad while recording, and then clicking through a transaction to mimic a user. QALoad will record all sent and received HTTP and SSL calls using the Script Development Workbench's Web proxy and write the activity to a capture file.

After recording, the capture file must be converted to an editable, C-based script. This is the point where Visual Scripting differs from a regular WWW script. By setting a single option before converting the capture file to an editable script, you can turn your capture file data into a Visual Script that allows you to view the actual Web pages you recorded in a browser-like interface, where you can manipulate the transaction and easily insert variable information into your script without directly editing a line of code.

To record a Visual Script:

1. Open a WWW session in the Script Development Workbench.
2. Click **Options>Record**. The WWW Record Options dialog box opens. [Set any relevant options.](#)
3. Click **OK**.
4. For convenience, set conversion options now also. (You can set conversion options after recording your transaction, if you prefer, or even change pre-set options at any time after recording and then re-convert the capture file to apply the changes.):
 - a. Click **Options>Convert**. The Universal Convert Options dialog box opens.
 - b. Set any applicable options on the General Convert and WWW tabs.
 - c. On the WWW tab, select the **Enable Visual Navigator** check box, which will ensure your capture file is converted to a Visual Script later.
 - d. Click **OK**.
5. From the toolbar, click the **Start Record** button. QALoad launches your application and any proxies, if necessary, and begins recording calls.
6. Record the transaction.
7. When you are finished, click the **Stop Record** button. You will be prompted to save your capture file. By default, capture files (.cap) are saved in the directory `QALoad\Middlewares\WWW\captures`.
8. If you previously set options to prompt automatic conversion, your capture file will be converted to a Visual Script and will open automatically in the editor. For more information about setting up automatic conversion, see [Configuring the Script Development Workbench](#).

If not, you will be prompted to save and name your file. When you are finished, click OK.

9. Click the Script Development Workbench's Captures tab and locate the capture file you just saved.
10. Right-click the file and choose **Convert**. QALoad will convert your capture file to a Visual Navigator script and open it in the editor.

Visual Navigator files

When you create a script using Visual Navigator, QALoad saves important information about your script in the following files. These files are saved in the directory `\Compuware\QALoad\Middlewares\WWW` in the subdirectories `\Scripts` and `\Captures`. Some of these files can be modified, and can be opened from the Script Development Workbench's Workspace pane, if necessary.

Filename	Description
----------	-------------

Files Generated From Recording	
<code><filename>.cap</code>	A file containing all of the requests and responses that were recorded.
<code><filename>.rfd</code>	Replies to subrequests, which mostly consist of images, style sheets, and javascripts. This data is used to visually recreate the pages as they appeared when recording.
Files Generated From Conversion to a Visual Script	
<code><filename>.vistree</code>	Contains most of the elements of the Visual Navigator tree, including any elements that you modify later or add to your script.
<code><filename>.VisHtml</code>	Contains the HTML pages of all the main requests as well as images, stylesheets, and other subrequested pages. This data is used to visually recreate the pages as they appeared when recording.
<code><filename>.VisXml</code>	Contains any XML/SOAP information that was recorded.
<code><filename>.cpp</code>	A C++ representation of your script.

Inserting script items

Inserting tree items

You can insert a number of script items into your converted script using the Visual Navigator menu accessed from the Script Development Workbench main menu, or by right-clicking in the tree-view (left pane).

From the main menu, choose Visual Navigator>Insert Tree Item and then choose the item to insert.

From the tree-view, right-click and choose Insert, and then choose the script item to insert.

Most of the inserted items can be moved up and down the tree using the Up/Down arrows in the form-view (bottom pane) for that item. You can also delete an item highlighted in the tree-view by choosing Delete Tree Item from the menu.

The following script items can be inserted from the Visual Navigator menu.

- Extract String
- Cookie
- Http Header
- Content Check
- CGI Parameter
- Synch
- IP Spoof
- Read Datapool
- Checkpoint pair
- Increment Variable/Decrement Variable/Reset Variable

Debug Print Comment

Inserting cookies into a script

Cookie items can be added directly to the Html Page item they apply to, under the Action item (for example, a Click on Link item).

To insert a cookie item:

1. In the Visual Navigator tree-view (left pane), navigate to the Html Page item requiring the cookie and then click on it to select it.
2. From the menu, choose **Visual Navigator>Insert Tree Item>Cookie**. A Cookie form-view opens in the bottom pane.
3. In the **Name** field, type a name for the new Cookie or click **var...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
4. In the **Value** field, type a value for the new Cookie or click **var...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
5. Click **Save** to save your changes.

The Cookie item is added to the script for the selected Html Page item.

Inserting HTTP headers into a Visual Navigator script

HTTP headers can be inserted under the Common Http Headers tree item.

To insert a new Http Header item:

1. In the Visual Navigator tree-view (left pane), navigate to the **Common Http Headers** script item, and then click on it.
2. From the menu, choose **Visual Navigator>Insert Tree Item>Http Header**. An **Http Header** form-view opens in the bottom pane.
3. In the **Name** field, type a name for the new header or click **var...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
4. In the **Value** field, type a value for the new header or click **var...** to access the Select Variable dialog box where you can select a value from a datapool file or create a variable for this field.
5. Click **Save** to save your changes.

The header item is added to the script, and will be used for all requests at playback unless it is overwritten by a header with the same name underneath an individual request action.

Inserting a datapool file into a Visual Navigator script

You can quickly add an existing datapool file to your script from the Visual Navigator tree-view.

To insert a datapool file:

1. Right-click anywhere in the tree-view. From the popup menu that opens, choose **Datapools and Variables**.
2. Click **Insert Datapool File**.

3. If your script doesn't yet include a central datapool file, the Choose Datapool Type dialog box will open, where you can designate whether you want to insert your datapool file as a central or local datapool. If your script already includes a central datapool, the Open dialog box will open.
4. From the Open dialog box, navigate to the datapool file to add to your script. Datapool files are normally located in the directory `\Compuware\QALoad\Datapools`.
5. Highlight the appropriate file and click **Open**.

The file you selected is added to your script and appears listed in your Datapools and Variables dialog box, where you can edit or delete it as necessary. If you expand the view of that file in the Datapools and Variables dialog box, you can see the variables (columns) saved in that file. The names for those columns are stored in the datapool file as a comment line. If they are not named, Visual Navigator will assign them the default name `Var#`.

Using datapools and variables with Visual Navigator

Variablizing a Visual Script

This topic describes several methods of working with variable data in a script.

Variablizing a string

Suppose you are testing a site where users must register themselves with a server. To register, they must type a name, password, address, and so on into a form. The resulting Visual Script would contain a Fill In Form item in the tree-view under the appropriate Page item. Clicking on a form item would open the item's form-view in the bottom pane, which might look similar to the following graphic:

Name	name
Value	Kim Walker

In this example, the edit box required the user to type their name. Assume that a second edit box required the user to type their password. Each edit box on the form resulted in a separate item under the Fill In Form tree item. At runtime, you might want to variablize the name and password edit boxes, so that a different user name and corresponding password is entered for every transaction. One way of variablizing these two fields is to create a datapool file containing two variables (columns) named Name (user ID) and Password, populated with valid user names and passwords.

To variablize the name edit box:

1. Highlight the text in the form-view. In this example, that would be *Kim Walker*.
2. Right-click on the highlighted text and choose **Substitute with Variable**, or simply click on the **var...** button.
3. On the Datapools and Variables dialog box that opens, click on the Name (user ID) variable and click **OK**.
4. The **Value** field in the form-view for that particular field will look something like the following:

Name	name
Value	{\$User name:User names and Passwords\$}

At run time, this form field will be filled in with a value from the User name column of the datapool file named User names and Passwords.

Working with an inserted variable

Here are some hints for working with variables:

- ! Clicking anywhere on an inserted variable selects the entire variable.

- ! You can delete a variable by clicking on it to select it, and then pressing the Delete key on your keyboard. Alternately, you can right-click on a variable and choose Delete Variable Reference from the menu to delete it.
- ! Rename a variable easily by double-clicking on it to open the Datapools and Variables dialog box with that variable highlighted. You can easily rename the variable from there.
- ! You can restore the contents of an edit box to whatever its contents were immediately after being converted by right-clicking on it and choosing Revert to Original String... from the menu. This will automatically delete any variables that have been inserted since the most recent conversion.

Mixing regular text with variables

An edit box can contain a combination of regular text and any number of variables, depending upon how complicated the string needs to be.

Using local variables

Another method of variablizing data, for instance the user ID and password describe previously, is to use local variables to create a unique name and password on each pass through the transaction loop with a format of User1 and Pass1, User2 and Pass2, and so on.

You could accomplish this using a combination of regular text and a local variable:

1. In the edit box for the User ID, type: `User`.
2. Then while the cursor is still at the end of the text you just typed, click the **var...** button to open the Datapools and Variables dialog box.
3. On the Datapools and Variables dialog box, create a local variable by clicking the **New Variable** button. Name the variable User Number. In the **Initial Value** field, assign the variable an initial value of 0 (zero).
4. Click **OK**. The **Value** field in the form-view will now display this value: `User{$User Number$}`

At run time, QALoad will create a value for that field that will be a concatenation of `User` and the value of the local variable named User Number.

5. Assume that the value should be 1 (one) for the first transaction, 2 (two) for the second transaction, and so on. To accomplish that, the value of User Number must be incremented before each iteration of the transaction. To increment the value before each transaction, insert an Increment Variable item before the Page item that uses the User Number variable.

Variablizing with a random value

Another method of variablization is to insert or substitute selected text with a Random Value tag.

1. Select text or position your cursor and then click the **var...** button.
2. On the Datapools and Variables dialog box, click the **Random Number** button. The Random Number Tag dialog box opens.
3. Type values in the **Lower** and **Upper** fields to specify a range from which the number should be drawn.
4. Click **OK**. The Value field in the form-view will now display a value similar to the following: `User{$Random:1:500$}`
5. At run time, this variable will create strings similar to the following: User17, User394, and so on.

To modify a range, double-click on the Random tag to open the Random Number Tag dialog box where you can adjust the ranges as needed.

Variablizing with a VU number

Another method of variablization is to insert or substitute selected text with a virtual user number tag.

1. Highlight the text to replace or position your cursor in a field and then click the **var...** button.

2. On the Datapools and Variables dialog box, click the VU Number (Absolute) or VU Number (Relative) button. The Value field in the form-view will be filled in with something similar to the following: `{SVU NUM ABS$}` or `{SVU NUM REL$}`.

At run time, the VU Number tag will be substituted with the number of the particular virtual user that is running this script. The string created will look similar to this: User45, User187, and so on.

Find/replace a variable

Visual Navigator has a Find/Replace feature that allows you to quickly find occurrences of strings within the tree-view and replace them with other strings. For example, you could find all occurrences of Smith and replace them with the datapool variable `{Last Name:User Info$}`. For details about the find/replace feature, see [Visual Navigator's Find and Replace feature](#).

Datapools and variables

Datapools and variables can be added or modified by several methods. To simply create, delete, or modify datapool files and variables at any time while a script is open in the editor, choose Visual Navigator>Datapools and Variables from the menu to access the Datapools and Variables dialog box.

Alternately, the same dialog box will open automatically whenever you are asked to choose a variable or datapool file while working with the script, allowing you to create the variables you need on-the-fly.

Data that can be variablized is denoted in the form-view (bottom pane) by the var... button. Clicking the var... button will open the Datapools and Variables dialog box.

For more information, click one of the following links:

[Creating/maintaining datapools](#)

[Adding variables](#)

Creating and editing datapools

The following sections provide step-by-step instructions for creating a new datapool file, importing a datapool file, editing a datapool file, and inserting a datapool file into a script.


To create a new datapool file:

1. With a script open, choose **Visual Navigator>Datapools and Variables** from the menu.
2. On the Datapools and Variables dialog box, click the **New Datapool File** button. If this script doesn't yet have a central (Conductor-based) datapool assigned, the Choose Datapool Type dialog box opens for you to specify whether the datapool you are about to create should be central (Conductor-based) or local (Player-based). Make your selection and click **OK**.
3. On the Create New Datapool File dialog box, specify a name for the new datapool and the number of rows and columns it should have.
4. Click **OK** to create the datapool file and be returned to the Datapools and Variables dialog box.
5. The Datapools and Variables dialog box now displays any datapool files assigned to the script, including the one you just created. Click **OK**.
6. In the tree-view, scroll to the Datapool Files section of the script and click on the name of the datapool file you just created.
7. In the form-view (bottom pane) type values into the datapool fields to be used at test time.
8. When you are finished, click **File>Save** to save your changes to the script.

To import a datapool file:

1. Copy your datapool file, which must be in .csv or .dat format, to the `Datapools` directory of the QALoad installation.
2. Insert the datapool into a script as described below.

To edit a datapool file:

 **Note:** Because it is possible for more than one script to use the same datapool file, care should be taken when modifying a datapool file's contents. Doing so for one script may cause errors in other scripts that use the datapool file. Also, Visual Navigator datapools should be edited within the Visual Navigator, under Datapool Files in the tree-view, instead of with the Datapool Editor.

If you need to make changes to a datapool file, Compuware recommends that you make a backup copy of the file first. If a backup copy is not available, you may need to check all other scripts that use that datapool file and make appropriate changes.

1. With your script open, scroll to the Datapool Files section of the script in the tree-view (left pane).
2. Locate the datapool file you want to edit, and click on it to highlight it. A table opens in the form-view (bottom pane) displaying the contents of the datapool file.
3. Edit the datapool:
 - Edit a cell's contents — Click in the cell and type over the existing contents.
 - Insert/delete columns or rows — To insert a row or column, right-click on a row or column header and choose **Insert Row** or **Insert Column** from the menu. Then choose whether to insert the new item before or after the selected item, and type the number of items to insert. To delete a row or column, right-click and choose **Delete Row** or **Delete Column** from the menu. (Press **SHIFT** to select multiple contiguous items or **CTRL** to select multiple non-contiguous items.)
 - Rearrange columns or rows — Select one or more columns or rows by clicking on the headers (press **SHIFT** first to select multiple items). Drag the rows or columns to a new position. As you drag them, a thin red line will indicate where the selection will be moved to.
 - Rename a column header (variable) — Click on the column name, and then type the new name in the Variable Name field.
4. Save your changes by clicking **File>Save**.

To insert a datapool file into a script:

1. With your script open, scroll to the Datapool Files section of the script in the tree-view (left pane).
2. From the menu, choose **Visual Navigator>Datapools and Variables**.
3. On the Datapools and Variables dialog box, click the **Insert Datapool File** button.
4. If the script doesn't yet have a central (Conductor-based) datapool assigned, the Choose Datapool Type dialog box opens. Select the option for the type of datapool to insert and click **OK**.
5. On the Open dialog box, navigate to the datapool file to use and click the **Open** button.
6. The file you selected will be added to the list of datapool files on the Datapools and Variables dialog box. Click **OK** to save you changes and be returned to the editor.

Inserting a datapool file into a Visual Navigator script

You can quickly add an existing datapool file to your script from the Visual Navigator tree-view.

To insert a datapool file:

1. Right-click anywhere in the tree-view. From the popup menu that opens, choose **Datapools and Variables**.

2. Click **Insert Datapool File**.
3. If your script doesn't yet include a central datapool file, the Choose Datapool Type dialog box will open, where you can designate whether you want to insert your datapool file as a central or local datapool. If your script already includes a central datapool, the Open dialog box will open.
4. From the Open dialog box, navigate to the datapool file to add to your script. Datapool files are normally located in the directory `\Compuware\QALoad\Datapools`.
5. Highlight the appropriate file and click **Open**.

The file you selected is added to your script and appears listed in your Datapools and Variables dialog box, where you can edit or delete it as necessary. If you expand the view of that file in the Datapools and Variables dialog box, you can see the variables (columns) saved in that file. The names for those columns are stored in the datapool file as a comment line. If they are not named, Visual Navigator will assign them the default name `Var#`.

Naming variables

When you first create a datapool file, the included variables are automatically assigned the default names `Var1`, `Var2`, `Var3`, etc.

QALoad allows you to rename those variables with meaningful names that can even include spaces. This makes it much easier to work with datapools. For example, you could name a datapool variable something logical like `City`, rather than trying to remember that `Var4` in your datapool is the `City` variable.

Renaming variables

You can quickly and easily rename local or datapool variables from the tree-view. Simply highlight the variable under the Datapool Files or Variables tree-view item, and then change the variable name in the resulting form-view (bottom pane).

You can also edit from the Datapools and Variables dialog box. To access it, right-click anywhere in the tree-view and then choose `Datapools and Variables` from the shortcut menu. Highlight the variable to rename and click the `Rename` button.

Adding a variable

You can quickly add local or datapool variables to your script from the Visual Navigator tree-view. A local variable can be substituted wherever variables can be used. You can insert `Increment Variable`, `Decrement Variable`, and `Reset Variable` items into the tree-view to manipulate the value of any variables.

To add a variable:

1. Right-click anywhere in the tree-view. From the shortcut menu, choose **Datapools and Variables**.
2. In the Datapools and Variables tree-view, click on the Variables item to add a local variable, or click on a specific datapool file to add a new datapool variable.
3. Click the **New Variable** button.
 - If you are adding a new local variable, then an unnamed variable will be added to your list of local variables. You can rename the new variable.
 - If you are adding a datapool variable, a variable with the default name (`Var#`) will be added to your datapool file. You can rename the new variable.
4. If you added a local variable, type the variable's value in the **Initial Value** field.
5. Click **OK**.

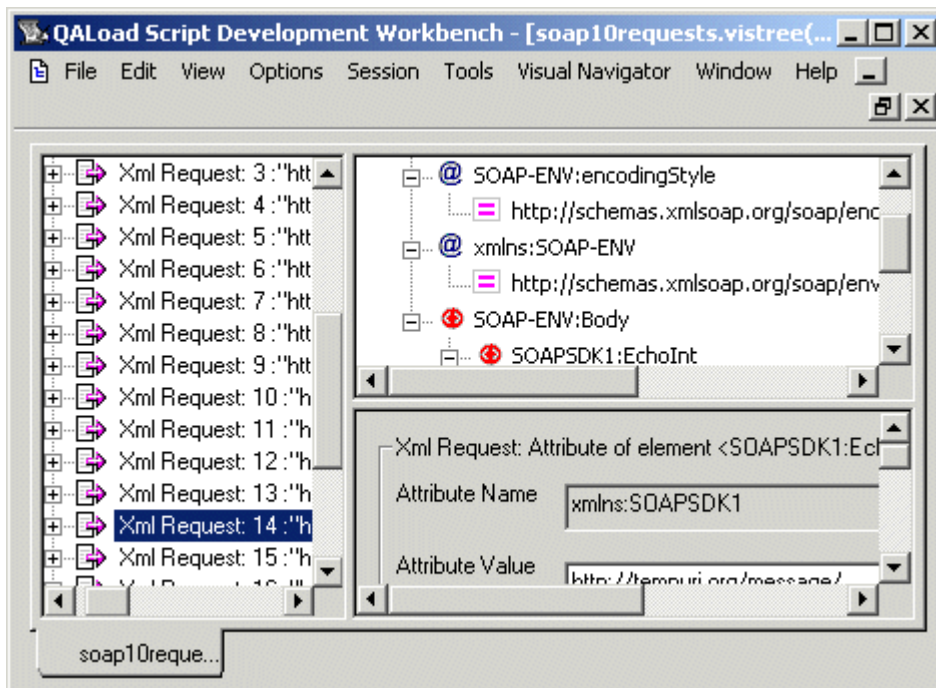
XML support

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request, or an HTTP POST request with an XML document as the post data, then the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from the **View** menu.



XML requests

When an HTTP request is for an XML document, either in the form of an HTTP GET request, or an HTTP POST request with an XML document as the post data, then an XML Request tree item will be displayed in the tree-view (left pane). The form-view (bottom pane) for an XML Request item includes the following fields:

Reply Status: The reply status code. For most pages which were returned correctly this will be 200 OK.

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

Checkpoint Name: If the page has a title, then it will be used as the checkpoint name. If not, the word Checkpoint will be used. To make sure all checkpoint names are unique, QALoad will add a number to the beginning of the checkpoint name.

XML Request sub-items

An XML Request item can contain the following sub-items.

XML Reply

The URI of the document returned as a result of the XML request. XML data corresponding to the reply is displayed in the browser-view.

HTTP Headers

If a header exists under an action item then it will be sent for that request only. If the header has the same name as one of the common headers, then it will override the common header for this request only. The form-view (bottom pane) for an HTTP header lists its name and value. Because there is no XML data recorded for a header, the browser-view remains empty. It is possible to insert additional HTTP Headers.

Cookies CGI Parameter

The Cookies tree item will contain a list of Cookie items that were sent in the header of the request that this item made while being recorded. Cookies are added automatically by the browser based on the URI that is being requested. They are either set as a result of the previous reply (server returned a Set-Cookie command), or they are set by JavaScript contained in the previous reply. The form-view for a cookie item lists its name and value. Because there is no XML data recorded for a header, the browser-view remains empty.

XML document-view

When you click on an XML Request item in the tree-view (left pane) the right pane becomes a document-view displaying a tree-view of details about the XML document requested or returned as the result of an XML request. Each individual XML item appears as a node in the XML document tree. XML elements can have child elements and these appear as child nodes of the XML element. Attributes of an element appear as child nodes of the element, with the attribute value appearing as a child of the attribute name.

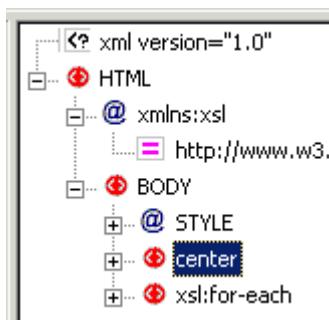
What if no XML data is associated with a request?

If there is no XML document associated with the XML request (for example, an HTTP GET) a message indicating that there is no XML to be displayed is shown in the XML document view.

How does the document-view relate to the form-view?

Selecting an item in the XML document tree will display the form-view details corresponding to that XML element type in the bottom pane.

Following is an example of XML request data displayed in a portion of the XML document-view:



XML form-view

When an XML request is displayed in the document-view (top pane) — as a result of an XML request item or XML reply child item being selected in the Visual Navigator tree-view — you can click on items in the document-view to display information about each in the form-view (bottom pane). If no XML item is selected in the document-view, the XML Page form-view will be displayed instead. For XML items, the form view display options depend on two things:

- ! what type of XML item is selected in the Visual Navigator tree-view (left pane): an XML request or an XML reply
- ! what type of XML item is subsequently selected in the XML document-view (top pane).

When an XML item is selected in the XML document-view, the value of that XML item is displayed in an edit box in the form-view. Some values — attribute values and text values — can be edited or variablized (that is, substituting one or more variables for the value in an XML request or selecting the return value from an XML reply item to be received by a variable for later use in the script). Text items, which are values between element tags, and attribute values represent volatile items in an XML document structure, used for passing values to and from Web Services, for example.

The following tables list the possible actions for XML items displayed in a form-view. Valid actions are determined by the XML item type and whether the item is from an HTTP POST request or from an HTTP reply.

In the following tables:

- ! If an item is *editable*, the value in the form-view can be changed and the new value will be used during replay.
- ! If a value can be *variablized*, a variable can be substituted for all or part of the value. The variable's value will be placed in the variable's location at replay. An example is a value received from an item from a previous XML document reply.
- ! If a variable can *receive* a replay value, the return value for the item can be placed into a selected variable during replay. The variable can then be substituted for an input value in a later XML request.

XML Request Items		
XML Request Item	Editable?	Can the Value be Variablized?
Declaration	No	No
DTD (Document Type Definition)	No	No
PI (Processing Instruction)	No	No
Comment	No	No
Element	No	No
Attribute (Name)	No	No
Attribute (Value)	Yes	Yes
Text	Yes	Yes

XML Reply Items	
XML Request Item	Can Variable Receive Replay Value
Declaration	No
DTD (Document Type Definition)	No
PI (Processing Instruction)	No

Comment	No
Element	No
Attribute (Name)	No
Attribute (Value)	Yes
Text	Yes

Visual Navigator's Find and Replace feature

Visual Navigator has an enhanced Find/Replace feature that allows you to find occurrences of strings within the tree-view, allowing you to quickly locate and/or replace text. For example, you could find occurrences of Smith and replace them all with the datapool variable {\$Last Name:User Info\$}.

Access Find and Replace from the Edit menu, or by pressing **Ctrl+F**.

Find tab

Find what: Type the string to search for, or click the **var...** button to select a variable to search for. The appropriate tree item will be selected and the found text will be highlighted in the form-view (bottom pane).

Case Sensitive: Select if Visual Navigator should only find strings with case matching that of the string to locate.

var...: Click to access the [Select Variable dialog box](#), where you can choose a local variable or datapool variable to search for.

Find Next: Click to find the next occurrence of the string.

Cancel: Click to cancel the search and close the dialog box.

Replace tab

Find what: Type the string to search for, or click the **var...** button to select a variable to search for. The appropriate tree item will be selected and the found text will be highlighted in the form-view (bottom pane).

Replace with: Type the string to replace the search string, or click the **var...** button to select a variable to replace the search string. The replacement string can be a combination of regular text and one or more variables.

var...: Click to access the [Select Variable dialog box](#), where you can choose a local variable or datapool variable to search for.

Case Sensitive: Select if Visual Navigator should only find strings with case matching that of the string to locate.

Allow read-only fields to be changed: Select to replace text in read-only fields without a confirmation dialog box. If this option is not selected, a confirmation dialog box appears for each occurrence of a string in a read-only field from which you can choose whether to replace the field.

Replace: Click to replace the string in the Find what field with the string in the Replace with field.

Replace All: Click to replace all occurrences of the string in the Find what field with the string in the Replace with field. Matching strings that are found in read-only fields will not be replaced.

Find Next: Click to locate the next occurrence of the search string.

Cancel: Click to cancel the search and close the dialog box.

[EasyScript for WWW](#)

Configuring a Web browser (WWW)

Before you record the WWW requests your Web browser makes, you must configure the browser to use QALoad's proxy server.

To configure a Web browser:


1. Start your Web browser.
2. Specify proxy settings:
 - In the field designated to specify the address of the proxy server, enter the machine name where QALoad Script Development Workbench is installed.
 - In the Port field, enter the port(s) that you specified on the Script Development Workbench's WWW Record Options wizard (Capture Ports fields).
3. Click **OK**.

Streaming media support

QALoad includes support for audio and video download testing of both Windows Media Player and RealOne Player and their supported media formats through a WWW session. When streaming media conversion is enabled and you record a transaction that calls streaming media, an additional command is inserted into your script which will request the media. You do not have to listen to or view the entire media you are requesting, you simply need to record its URL and ensure that the appropriate media player is installed on the Player machines that will play back the script. At run time, the script invokes your media player and requests the streaming media resource. Streaming media through a firewall or proxy server is not supported.

QALoad's streaming media support includes the following media player(s). The appropriate media player must be installed on the machine you are recording from as well as any QALoad Player machine that will be executing the script.

- ! **RealOne Player** — The media download is initiated by requesting a file that is a data type supported by the RealOne Player. Supported data types are RealAudio, RealVideo, RealText, RealPix, MP3, and SMIL. As a result, the [DownloadRPMedia](#) command will be inserted into the script at the appropriate point. At runtime, this command initiates and waits for completion of the download. RealOne Player scripts must be executed as process-based scripts.
- ! **Windows Media Player** — The media download is initiated by downloading a file with a content type of (audio|video)/(x-ms-asf|s-ms-asf) in the browser. Currently, only .asx files are supported. As a result, the [DownloadMediaFromASX](#) command is inserted into the script at the appropriate point.

 **Note:** QALoad does not support scripts that have both RealNetworks media and Windows Media in the same script. To test both types in a single load test, use a different script for each type.

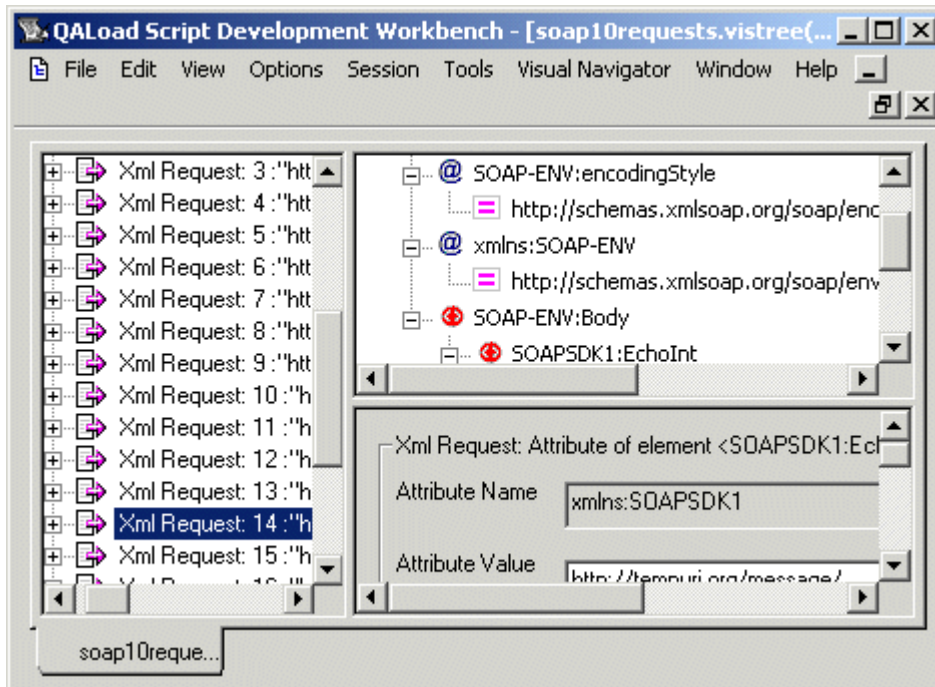
Please note that asynchronous calls may not be played back exactly as they were recorded. For example, if you click on a link in the browser while recording while the media is playing, during replay that link will not be requested until the media clip has finished being processed.

XML Support

QALoad's XML support is handled through the Script Development Workbench's Visual Navigator, which displays your script's HTTP or XML requests in an easy-to-use visually-based interface that offers you point-and-click script editing. Although XML is supported through the Visual Navigator, we recommend you read through this help topic as well as the Visual Navigator help topics to become familiar with the features that are unique to QALoad's XML support.

When an HTTP request is made for an XML document, either in the form of an HTTP GET request, or an HTTP POST request with an XML document as the post data, then the data is displayed in the three Visual Navigator panes as illustrated below. Click on a pane in the graphic for a description of its contents and functionality.

Note: To make the following graphic fit better in this help window, we've turned off the Script Development Workbench toolbars and panes that are not directly related to this help topic. You can hide/show many of the Script Development Workbench toolbars and panes using commands available from



DBCS Support in QALoad

QALoad supports load testing of Chinese, Japanese and Korean Web applications that use Double Byte Character Sets (DBCS). DBCS is a character set that uses two bytes (16 bits) rather than one byte (8 bits) to represent a single character. Some languages, such as Chinese, Japanese, and Korean, have writing schemes with many different characters and character sets that cannot be represented with single-byte characters such as ASCII and EBCDIC.

Note: DBCS support only applies to the WWW/SSL middleware. Currently QALoad only supports DBCS; there is no support for Web applications that host Bi-Directional (BiDi) characters (which includes Arabic and Hebrew languages).

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

- ! **Native Character:** converts the DBCS characters to their original characters in Chinese, Japanese, or Korean. Native character support is only possible when using a native operating system (OS) such as load testing a Japanese Web application from a Japanese version of Microsoft Windows. While QALoad does not support different DBCS characters within the same script, it does support a script containing the DBCS native character set and ASCII/English. Native character support is used within test scripts, error messages (generated through system commands that use native characters) and timing files, making script editing easier and timing file analysis quicker.
- ! **Encoding:** encodes all DBCS characters into a sequence of printable characters regardless of the language and exact character set in use. Encoding support is used when load testing multiple language sites from the same OS, or when load testing a DBCS site from that of another DBCS platform. For example, testing a site with Japanese characters from a Korean or English/ASCII OS. The encoding option is used when native character support cannot be used or when script portability between different DBCS language OS is required.

WWW conversion options

Form field as comments: Select this check box to include a comment block in your script that shows each valid field that is encountered after a CGI form has been requested. This option is not available if the Visual Navigator is enabled. Following is an example of a form field comment block:

```
/* Form:1 text Name: name, Value: , Desc: */
/* Form:1 text Name: e-mail, Value: , Desc: */
/* Form:1 text Name: Address, Value: , Desc: */
/* Form:1 text Name: city, Value: , Desc: */
/* Form:1 text Name: state, Value: , Desc: */
/* Form:1 text Name: zip, Value: , Desc: */
/* Form:1 check box Name: echo, Value: , Desc: Echo a copy of HTML page to email */
/* Form:1 radio Name: test, Value: capture, Desc: Capture */
/* Form:1 radio Name: test, Value: replay, Desc: Replay */
/* Form:1 hidden Name: hidden, Value: This rocks!, Desc: */
```

Anchors as comments: Select this check box to include a comment block in your script that shows all the anchors encountered in a requested HTML document. If it is not selected, no anchors will be included in the script. This option is not available if the Visual Navigator is enabled. The following is an example of an anchors comment block:

```
/* Anchors:'http://playback1/standard.html' 'Standard Example' */
/* Anchors:'http://playback1/forms.html' 'Forms Example' */
/* Anchors:'http://playback1/dynamic.html' 'Dynamic HTML Example' */
/* Anchors:'http://playback1/cgi.html' 'CGI Example' */
/* Anchors:'http://playback1/cookies.html' 'Cookies Example' */
/* Anchors:'http://playback1/ssl.html' 'SSL Example' */
/* Anchors:'http://playback1/javascript.html' 'Java Script Example' */
```

Client Image Maps as Comments: If selected, the command `DO_GetClientMapHREF` will be inserted into your script followed by a comment block denoting client image maps. This option is not available if the Visual Navigator is enabled.

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

/* Client Map:1 Region:2 HREF: http://www.ethnicgrocer.com/eg/cp/cr.jsp?FOLDER%3C */
/* %3Efolder_id=16991 &ASSORTMENT%3C%3Eeast_id=153827&site=EG&bmUID= */
/* 1005685611375&WebLogicSession=01GLayxDvTfjLthL65xY6X1cdQIVdCeJwCT8wm */
/* D4PLs29z9H7WC wxlkr8f21K1aKoLSMI4Hml6o3|6518173674514870389/ */
/* 167838850/5/80/80/443/443/-1|-8250053020002903791/167838870/5/80 */
/* /80/443/443/-1|6518173674514872679Client Map:1 Region:3 HREF: */
/* http://www.ethnicgrocer.com/eg/cp/cr. */
```

You can search for `DO_GetClientMapHREF` in the script to help locate the `DO_Get` and `DO_SetValues` in the script, as well as where the array of `DO_GetClientMapHREF` is initialized and freed.

Debug comments: When this option is enabled, comments will be inserted into the converted script to denote replies from the server, anchors, and so on. This option is not available if the Visual Navigator is enabled. For example:

```
/* Received reply:
URL:<http://abcweb.anywhere.com/cafe/default.htm> */
```

Document Title Verification: Periodically, the HTML page title found in a load tested script does not match the HTML page title found in your recorded text. Select this check box to compare the HTML page title contents in your load tested script with the HTML page title contents in your capture file. You can enable the comparison based on prefix- or suffix-specified character match or entire string match.

Entire Document Title: Select this option to compare the entire length of the HTML page title.


Prefix: Select this option to compare the prefix (left-most specified characters) of the HTML page title. Specify the number of characters to match in the Characters to Match field.

Suffix: Select this option to compare the suffix (right-most specified characters) of the HTML page title. Specify the number of characters to match in the Characters to Match field.


Characters to match: After you select the Prefix or Suffix options, use this field to specify the number of characters to match from the HTML page title.

Baud Rate Emulation: Select this check box to download web pages and images at a rate representing the speed of connection, then enter a connection speed in the Baud Rate field. This enables you to simulate modem speed.


Refresh Timeout: Select this check box and type a time value in the field (in seconds) to compare the specified time value against a Web page's META Refresh value (e.g. `<META HTTP-EQUIV=Refresh CONTENT="10" ; URL="http://www.compuware.com/">`). If the META Refresh tag's CONTENT field value is less than the time value you specify, the page is treated as a redirected page. If the CONTENT field value is greater than the time you specify, the page is treated as a regular page.

 **Tip:** Select this option to avoid infinite loops in the script. Infinite loops can occur if a page refreshes periodically to update data.

Encode DBCS Characters: Select this check box to enable the encoding of captured data from Web applications containing Double Byte Character Sets (DBCS) such as Chinese, Japanese, or Korean. Enabling this option encodes all native characters and is used when native character support cannot be used. Data stays in encoded format throughout the load test: from capture, through convert and replay, to the analysis of the timing file. By leaving this option clear (default), DBCS native characters will be used within test scripts, error messages, and timing files, making script editing easier and timing file analysis quicker.

 **Note:** Native character support can only be used on the same DBCS language OS as the application under test. For further information on DBCS Support, see [DBCS Support in QALoad](#).

Enable Visual Navigator: Select this check box to enable [Visual Navigator](#), producing a visually-oriented script rather than the standard QALoad C-based script.

 **Note:** When you select this option, some WWW conversion options are not available, such as commenting options, and some automatic playback options.

Advanced button: Accesses the [WWW Advanced](#) dialog box, allowing you to set advanced conversion options.

WWW recording options

Automatic Startup of Internet Explorer: Select this option to have QALoad automatically launch the browser and configure the proxy and port entries for the browser and the QALoad Script Development Workbench. If you do not select this option, you must manually configure your proxy options before recording.

User Started Application: Select this option to configure the browser or application manually, and then set the appropriate options from those that follow.

Proxy Settings for User Started Application

Direct Connection: Select this option if you have a direct connection to the host from which you are recording.

Manual Proxy Configuration: Select this option to specify a proxy through which to connect to the host. Then, use the HTTP, Secure, and Exceptions fields to set proxy options.

HTTP: Type the address of the HTTP proxy server. This field has a 255 character limit. Then enter the port number in the Port field. QALoad accepts numbers from 0-65535.

Secure: Type the address of the secure proxy server. This field is only available if you are licensed to use EasyScript for Secure WWW. This field has a 255 character limit. Then enter the port number in the Port field. QALoad accepts numbers from 0-65535.

Exceptions: Type the addresses of any hosts for which QALoad should not use the specified proxies.

Proxy Automatic Configuration Script: Select this option if your proxy should use an automatic configuration script located on your local area network. Then type the script's URL in the URL field.

Advanced Options

Advanced Options: This area displays the settings for several options. To change any of these settings, click the Change Advanced Options button to open the Advanced Options dialog box.

WWW command reference

QALoad provides descriptions and examples of the various commands available for a WWW script. For details, refer to the Language Reference Help section for [WWW](#).

Advanced scripting techniques for WWW

Simulating variable IP addresses

While QALoad can simulate multiple virtual users from a single system, it generally does so using a single source IP address. In most testing situations this isn't a problem, but with a small set of HTTP-based applications, it may not be the best way to simulate real-life activity. For QALoad Player machines with more than one static IP address, QALoad can direct each virtual user to use a different source IP address. To accomplish this, a local datapool file containing a list of local static IP addresses must be created on each QALoad Player machine. When you enable IP spoofing in the QALoad Conductor, the QALoad Conductor instructs each QALoad Player to create the appropriate datapool file at run time. The QALoad Player 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

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

Creating a Datapool of IP Addresses


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

To create a datapool of IP addresses:

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

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

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

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

Handling error messages from the Web server

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

Handling error messages with response codes

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

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

Example

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

DO_Http("GET http://www.host.com/ HTTP/1.0\r\n\r\n");
if((error = DO_GetLastHttpError()) > 399)
{
    sprintf(errorString, "Error in response: %d\n", error);
    WWW_FATAL_ERROR("Request-host", errorString);
}
```

Handling error messages returned in an HTML page

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

Using DO_VerifyDocTitle to verify page requests

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

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

Example

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

Searching response text for error messages

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

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

Example

```

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

```

Simulating CGI requests

The following topics describe strategies for simulating CGI requests:

[CGI parameter encoding](#)
[CGI Get requests](#)
[CGI Post requests](#)
[CGI forms](#)

Simulating JavaScript

JavaScript is handled by the following process:

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

Supported objects

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

Supported properties

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

Evaluation errors

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

Example Web page

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

```

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

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

function scrollit_r21(seed)
{
var m1 = " Welcome to Compuware's QALoad homepage.";
var m2 = " Glad to see you.";
var m3 = " Thanks for coming. ";
var msg = m1 + m2 + m3;
var out = " ";
var c = 1;

if (seed > 100) {
seed--;
var cmd="scrollit_r21(" + seed + ")";
timerTwo=window.setTimeout(cmd,100);
}

```

QALoad 5.02

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

else if (seed <= 0) {
if (-seed < msg.length) {
out+=msg.substring(-seed,msg.length);
seed--;
var cmd="scrollit_r2l(" + seed + ")";
window.status=out;
timerTwo=window.setTimeout(cmd,100);
}
}

else {
window.status=" ";
timerTwo = window.setTimeout("scrollit_r2l(100)", 75);
}
}
}
}

</script>

<BODY onLoad="timerONE=window.setTimeout('scrollit_r2l(100)',500);">
<!-- End scrolltext -->

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

</BODY></HTML>
```

Example script

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

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

```
DO_InitHttp(s_info);

...
...

BEGIN_TRANSACTION();
DO_AutomaticSubRequests(TRUE);

...
...

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

...
...

END_TRANSACTION();
```

Executing a Visual Basic script

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

Executing a Java applet

Java applets are handled by the following process:

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

Example Web page

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

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

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

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

<br><br><br>

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

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

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

<br><br><br>

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

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

Example script

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

QALoad 5.02

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

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

```
DO_InitHttp(s_info);
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://www.host.com/java.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Java Example", TITLE);
/* Request: 2 */
DO_Http("GET http://www.host.com/LScrollText.class HTTP/1.0\r\n\r\n");
/* Request: 3 */
DO_Http("GET http://www.host.com/imagefader.class HTTP/1.0\r\n\r\n");
DO_Http("GET http://www.host.com/jak.jpg HTTP/1.0\r\n\r\n");
...
...
END_TRANSACTION();
```

Simulating frames

Frames are handled by the following process:

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

Example Web page

The following Web page contains four frames.

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

Example script

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

! A DO_Http call to retrieve the main page.

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

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

...
...

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

...
...

END_TRANSACTION();
```

Simulating cookies

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

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

Example Web page

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

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

<html>

...

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

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

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

Example script when Dynamic Cookie Handling is turned on

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

! Two CGI requests that return dynamic pages

! Cookies are handled by the replay engine

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

...
...

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

QALoad 5.02

```
/* 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 QALoad Script Development Workbench disables browser caching while recording, which means a page is always retrieved.

Requesting password-protected directories

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

Example script

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

The example script features the following elements:

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

```
BEGIN_TRANSACTION();
DO_BasicAuthorization("frank", "~encr~557A2549474E57444A");
...
...
DO_Http("GET http://www.host.com/access_controlled/secure.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of a Secured Page", TITLE);
...
...
END_TRANSACTION();
```

Example script

QALoad also handles Windows Domain Authentication (NTLM).

The example script features the following elements:

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

```
BEGIN_TRANSACTION();
DO_NTLMAuthorization("dom1\\frank", "~encr~557A2549474E57444A");
...
...
DO_Http("GET http://www.host.com/ntlm_controlled/secure.htm HTTP/1.0\r\n\r\n");
DO_VerifyDocTitle("Successful Test of a NTLM Page", TITLE);
```

...
...

```
END_TRANSACTION( ) ;
```

Using the WWW Convert Options dialog box

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

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

Advanced options:

- Cache
- Dynamic redirect handling
- Dynamic cookie handling
- Automatically process subrequests
- Persistent connections during replay
- Reuse SSL session ID
- Max concurrent connections
- Max connection retries
- Server response timeout
- HTTP version detection
- ActiveData
- IP spoofing
- Streaming media
- Hostnames as IP addresses
- Strip all cookies from request
- Traffic filters

EasyScript for Secure WWW

Overview


EasyScript for Secure WWW supports SSL/HTTPS requests when used in conjunction with the WWW middleware. This support must be purchased separately and is distributed in a separately-installed module.

Importing a client certificate from a Web browser (SSL)

You can import and convert a Client Certificate for any Web site you plan to visit.

To import a client certificate:

1. Start your Web browser.
2. From the browser, select the Client Certificate for the Web site you plan to visit.
3. Export the Client Certificate (.p12 or .pfx file) to a directory where you can access it using the Script Development Workbench.

 **Note:** When the browser prompts you to enter a password, **do not** enter a password. If you enter a password, QALoad cannot process the file.

4. Start a WWW Session in the QALoad Script Development Workbench.
5. Click **Tools>Maintain Certificates** to open the SSL Certificate Maintenance dialog box.
6. On the **Client Certificates** tab, click the browse button [...] to browse for the Client Certificate you want to convert. The Select the Exported Client Certificate to Convert dialog box opens.
7. Make sure **Files of Type** specifies P12 files (*.p12) or PFX files (*.pfx).
8. Select the appropriate Client Certificate and click **Open**. The path and file name of the selected Client Certificate appears in **Enter Certificate to Convert** on the Client Certificates tab.
9. On the **Client Certificates** tab, click **Convert**.
10. Click **Close** to exit the SSL Certificate Maintenance dialog box.

Creating a client certificate in QALoad (SSL)

This procedure assumes you have a WWW session active.

To create a client certificate:

1. From the **Tools** menu, select **Maintain Certificates** to open the **SSL Certificate Maintenance** dialog box.
2. On the Client Certificates tab, enter a name in the **Certificate Name** field.
3. Enter the number of certificates to create.
4. Click the **Create** button to create the QALoad Client Certificate. QALoad stores it in the QALoad\Certificates directory.
5. If necessary, configure your Web server to accept QALoad as the Certificate Authority. Refer to your Web server documentation for more information.

Creating an SSL Certificate Authority

Note that creating a new CA invalidates all previously created client certificates.

To create an SSL Certificate Authority:

1. Start a WWW session.
2. From the **Tools** menu, select **Maintain Certificates**.
3. Click the Certificate Authority tab.
4. Click the **Create** button to create a new Certificate Authority with the expiration date shown in the field.
5. Exit and re-start the Script Development Workbench.
6. After creating a new Certificate Authority, re-import the CA to your Web server and then create new Client Certificates.

Creating an SSL Server Certificate

To create an SSL Server Certificate:

1. Start a WWW session.
2. From the **Tools** menu, select **Maintain Certificates**.
3. Click the Server Certificate tab.
4. Click the **Create** button to create a new Server Certificate with the expiration date shown in the field.

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 `\Program Files\Compuware\QALoad\Certificates` directory to the same default directory on the Player machine.

SSL Commands

QALoad provides descriptions and examples of the various commands available for a secure WWW script. For details, refer to the Language Reference Help section for [SSL](#).

NetLoad


Using NetLoad

NetLoad is QALoad's suite of load generation scripts that allows you to simulate load conditions on your network using any of the following protocols:

- ! FTP
- ! HTTP
- ! PING
- ! LDAP
- ! POP3
- ! SMTP
- ! TCP
- ! UDP
- ! MExchange

NetLoad includes QALoad-provided scripts, which you can access from the Conductor to run in a test, for each protocol. You can customize the activity of the script by creating reusable datapools in the QALoad Script Development Workbench to use during testing. When you run a test, each virtual user will request a single datapool record. Once all the records have been read, the datapool file is rewound and the process starts again. You can use QALoad's components to run scripts and analyze the results as usual, or you can integrate your results with Compuware's ServerVantage product.

In short, NetLoad allows you to generate traffic on your network in a controlled manner and gather performance timings from the network. To facilitate testing under TCP/IP and UDP, NetLoad provides you with a server module to simulate server activity — allowing you to gather network timings without expending your actual server resources.

 **Note:** To use NetLoad for MExchange to test on Outlook 2000, you must ensure that CDO support is installed on your workstation before you continue. For instructions, see [Verifying CDO Support for MExchange](#).

For more information on the NetLoad Server modules, see [NetLoad Server Modules for TCP/IP and UDP](#).

NetLoad server modules for TCP/IP and UDP

If you are load testing a network running TCP/IP or UDP, you should use the appropriate NetLoad Server module to simulate server responses during your load test. This allows you to load your network and collect timings without expending your own server's resources. The NetLoad Server modules are only for use if you're testing on TCP/IP or UDP. You do not need to install the Server modules to test any other NetLoad-supported protocol.

You can install or copy the NetLoad Server modules to any Windows workstation on your network. After starting the appropriate Server module, you supply the QALoad Script Development Workbench with the

host name of the machine where the Server module is running and the port number that you specified when you started the Server module. When you are ready to run a test, start the Server module first. During the test NetLoad communicates with the NetLoad Server module, effectively loading the network. If NetLoad does not find the NetLoad Server module at the specified port—for instance if you mistyped the port number—the test fails (TCP) or fails to initiate (UDP).

Determining when to use the TCP server module


If you are going to send TCP packets using NetLoad, you must have a QALoad TCP Server module running on each machine that you are sending packets to. Copy the TCP Server module file, `NetloadTCPServer.exe`, to each machine that will be receiving packets and double-click on the file to start the TCP Server module.

Because the QALoad TCP Server module is a Windows-based program, you cannot use it to send NetLoad TCP packets to a UNIX machine.

Determining when to use the UDP server module

It is not necessary to have a QALoad UDP Server module running at the destination machine for NetLoad to successfully send packets to it; however, the Netload UDP Server can be useful to verify that the packets are being sent. To install the UDP Server module on a machine you are sending packets to, copy the program `NetloadUDPServer.exe` to that machine. Double-click the file to start the UDP Server module.

Since it is not necessary to have the UDP Server module running, you can send NetLoad UDP packets to both UNIX and Windows workstations.

 **Note:** If you are testing UDP in “broadcast” mode, it is not necessary to use the NetLoad Server module.

Installing the NetLoad Server module

If you are load testing a network running TCP/IP or UDP, the NetLoad Server module appropriate for your protocol must be running on a Windows workstation on your network before you start the test. The Server modules are installed automatically if you chose the option to install them during setup. However, once the Server module is installed on one workstation, you can install it on another workstation by simply copying the program from one workstation to another. The NetLoad Server modules are installed to the directory `\Program Files\Compuware\QALoad\Middlewares\NetLoad\Server`, and are named:

- ! **NetLoadTCPServer.exe** (for TCP/IP): If you are going to send TCP packets using NetLoad, you must have a TCP Server module running on each machine you are sending packets to. Because the TCP Server module is a Windows program, you cannot send NetLoad TCP packets to a UNIX machine.
- ! **NetLoadUDPServer.exe** (for UDP): It is not necessary to have a UDP Server module running on the machines you are sending UDP packets to. However, the UDP Server is useful for verifying that the packets are being sent. Since it is not necessary to have a UDP Server module installed on the destination workstations, you can send NetLoad UDP packets to UNIX machines.

Starting the NetLoad server module

You can configure and start the NetLoad server module from the Start menu.

If you are load testing a network running TCP/IP or UDP, the NetLoad Server module appropriate for your protocol should be running on a Windows workstation on your network before you start the test. The Server modules are installed with your QALoad product if you chose to install them during setup. If you are unsure if you should be using a NetLoad Server module, see [NetLoad server modules for TCP/IP and UDP](#).

To start the module:

1. Point to **Start>Programs>Compuware> QALoad >NetLoad**. Then click on the appropriate Server module: **TCP Server** or **UDP Server**.
2. When prompted, type the port number of the host machine and click **OK**.

3. On the QALoad NetLoad Server window, under the **Options** menu, select one of the following:
 - **Show Message Every Packet** — Displays a message, including byte size, after sending or receiving a packet.
 - **Show Message Every 100 Packets** — Displays a message every 100 packets listing the total number of packets received.

Starting a NetLoad session

You can start a NetLoad session from the workbench with an existing datapool file or a new one.

To start a session:

1. From the QALoad Script Development Workbench, choose **Session>NetLoad**.
2. Open an existing protocol datapool file or create a new one:
 - To create a new datapool file, choose **File>New**. The New NetLoad File dialog box opens.
 - To open an existing datapool file, choose **File>Open**. The Open NetLoad File dialog box opens.
3. Select the protocol you wish to test on and click **OK**. If you are opening an existing datapool file, navigate to the file and open it.
4. Enter or edit the appropriate datapool information in the **Workbook Pane**.

The QALoad Script Development Workbench allows you to have multiple files open at the same time.

Datapool files are located in the directory `\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts`.

Creating a NetLoad datapool

To create a NetLoad datapool:

1. From the QALoad Script Development Workbench, click **Session>NetLoad**.
2. Click **File>New** to open the New NetLoad File dialog box.
3. Select the protocol for which you wish to create a datapool file and click **OK**.

A grid opens in the Workbook Pane. Each row on the grid represents a single data record. The column headings indicate the appropriate field information to enter. Note that the actual fields in the grid vary by protocol.

4. Enter the appropriate information for your datapool file.

Some fields on the grid contain pull-down menus. To activate them, click anywhere within the field. Then make your selection from the menu that appears.

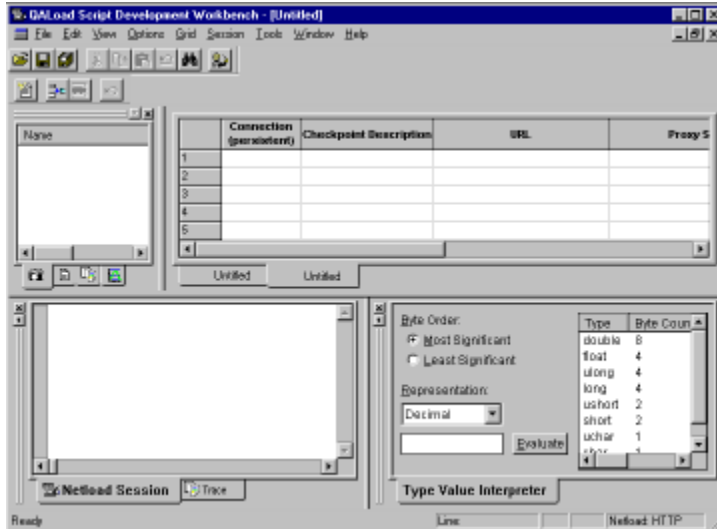
5. When you are finished, select **File>Save** to name and save the datapool file.

The datapool file is listed in the Workspace Pane Datapools tab. QALoad creates a script with the same name and lists it on the Scripts tab. Both files are saved to the `\NetLoad\Scripts` directory (for example, `c:\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts\datapool.dat`).

To enter datapool data:

1. From the QALoad Script Development Workbench, choose **Session>NetLoad**.

Click **File>New** to open the New NetLoad File dialog box. Select the protocol for which you wish to create a datapool file and click **OK**. A grid similar to the one shown below appears in the Workbook Pane. Each row on the grid represents a single data record. The column headings indicate the appropriate field information to enter. Note that the actual fields in the grid vary by protocol.



2. Enter the appropriate information for your datapool file. Note that some fields on the grid contain pull-down menus. To activate them, click anywhere within the field. Then make your selection from the menu that appears.
3. When you are finished, click **File>Save** to name and save the datapool file. Note that your datapool file is listed in the **Workspace Pane Datapools** tab. QALoad creates a C-based script by the same name and lists it in the Workspace Pane Scripts tab. Both files will be saved to your \NetLoad\Scripts directory (for example, c:\Program Files\Compuware\QALoad\Middlewares\NetLoad\Scripts\datapool.dat).
4. (Optional) Write a description of this datapool file for later reference by selecting **Options>NetLoad**. Once a description has been entered for a datapool file, you can review or edit the description any time the file is open by selecting **Options>NetLoad** again.

Editing a NetLoad datapool

You can edit the NetLoad datapool to make changes or additions to the file.

To edit a datapool:

1. With the appropriate NetLoad protocol session open, open the datapool by choosing **File>Open** and navigating to it, or select it from the **Workspace** tab **Datapools** tab.
2. Make any changes or additions to the file.
3. To delete an entire record (a single row), click its row number and select **Grid>Delete Row(s)**.
4. To insert a new record (a single row) above an existing record, click a row number and select **Grid>Insert Row**. NetLoad inserts a blank row above the selected row.
5. Save any changes to the file by selecting **File>Save**.

Adding or editing a NetLoad datapool description

You can add a meaningful description, or edit a previous one, for any NetLoad datapool.

To edit a description:

1. With a datapool file open, select **Options>NetLoad**.
2. Enter a description for the current datapool file.

UNIX


Installing UNIX Players

The necessary UNIX Player software is distributed with your QALoad Windows installation.

 **Note:** For updated UNIX system requirements and installation procedures, please refer to the QACenter Performance Edition Installation and Configuration Guide. You can access this guide by clicking **Start>Programs>Compuware>QALoad>Documentation>Installation and Configuration Guide**.

Transferring scripts to a UNIX Player

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

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


Transferring a Script

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

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

Setting up for DB2 playback

As with all QALoad middleware support on UNIX, DB2 UNIX support is replay only. QALoad does not support recording scripts from a UNIX environment. QALoad assumes that the DB2 environment is working prior to installation of QALoad .

 **Note:** To run DB2 load tests on AIX with 10 or more virtual users in thread-based mode, you must set the DB2 environment variable `EXTSHM` to `ON` to work around a memory handling problem in DB2.

To use EXTSHM with DB2:

1. Before starting the client application, type the following command:

```
export EXTSHM=ON
```
2. When starting the DB2 server, type the following commands:

```
export EXTSHM=ON
db2set DB2ENVLIST=EXTSHM
db2start
```

For information about setting up your UNIX Player installation, refer to the QACenter Performance Edition Installation and Configuration Guide. You can access this guide by clicking [Start>Programs>Compuware>QALoad >Documentation>Installation and Configuration Guide](#).

Testing with QARun

Creating a QARun script

To create a QARun script, you insert any number of QARun transactions (QARun scripts) into a QALoad template script accessible from the QALoad Script Development Workbench. The template script is a simple QALoad script that can be compiled and run; however, it contains no functionality until you insert the QARun transactions appropriate for your testing needs. QALoad provides two methods for inserting QARun transactions: automated, and manual.

Using the [automated method](#), you enter information in the QALoad Script Development Workbench about the QARun transactions to use and then let QALoad generate the test script using the information you provided. This method is fast and efficient when you know exactly which QARun scripts to use and where they are located.

The [manual method](#) allows you to open a copy of the QALoad template script and insert transactions and commands manually. You may want to use this method if you suspect you may need to edit your script while you're creating it.

Automatically creating a QARun script

To automatically create a QARun script:

1. From the QALoad Script Development Workbench, click **Session>QARun** to start a QARun scripting session.
2. Click **Session>Generate Script**. The Create New QARun Execution Script dialog box opens.
3. In the **Login String** field, select or type a valid username and password to access your installation of QARun.
4. In the **Environment** field, select the appropriate QARun environment.
5. In the **QARun Script Name** field, enter the name of the QARun transaction to insert, or select it from the list, which contains a record of the last five QARun script names you entered.

Although you can enter a script name from any database, when the test is actually running and QALoad invokes QARun, QARun attempts to retrieve that script from its default database. Therefore, in the QARun program installed on the Player, you should designate a default database that contains the script(s) you want to run.

6. Select the **Automatically Include Checkpoint** check box if you want QALoad to automatically insert a checkpoint into the script after this QARun transaction.
7. In the QALoad Script Name field, enter a name for this QALoad script. To write over an existing script, click the **Browse** button to the right of this field and select a script from the list of available scripts.
8. To add additional QARun transactions to this script, click the **Add Another Script** button and repeat Steps 3–6 for each additional transaction.
9. When you are finished, click the **Create Script** button. The QALoad script is saved in the directory `\Program Files\Compuware\QALoad\Middlewares\QARun\Scripts`, and the script opens in the script editor.

10. To compile the script for testing, click **Session>Compile**.

Manually creating a QARun script

You can manually insert QARun commands or scripts into a QALoad script to compile.

To manually create a script:

1. From the QALoad Script Development Workbench, select **Session>QARun** to start a QARun scripting session.
2. Select **Session>New Template** to create a new script from the QALoad template script.
3. In the Choose Script Name dialog box, enter a name for the new QALoad script and click **OK**. The script is saved in the directory `\Program Files\Compuware\QALoad\Middlewares\QARun\Scripts`, and the script opens in the script editor.
4. Edit the script as necessary:
 - ! You can manually enter any transactions or scripting commands directly in the script.
 - ! You can insert a QARun transaction by positioning the cursor on the appropriate line and selecting **Session>Insert>Transaction**. On the **Insert a QARun Transaction** dialog box that opens:
 - ! In **Login String**, select or type a valid user name and password to access your installation of QARun.
 - ! In **Environment**, select the appropriate QARun environment.
 - ! In **QARun Script Name**, enter the name of the QARun transaction to insert, or select it from the list, which contains a record of the last five QARun script names you entered. Note that you can enter a script name from any database; however, when the test is actually running and QALoad invokes QARun, QARun will attempt to retrieve that script from its default database. Therefore, in the QARun program installed on the Player, you should designate a default database that contains the script(s) you want to run.
 - ! When you are finished, click **Insert** to insert the script you just created into the QALoad script.
5. When you are finished, save any changes.
6. To compile the script for testing, select **Session>Compile**.

Troubleshooting

The Default Session Prompt didn't open?

If the Default Session Prompt fails to open when you start a middleware session, then default session checking was previously disabled. Do the following to enable default session checking:

1. From the **Options** menu, choose **Workbench**. The Configure QALoad Script Development Workbench dialog box opens.
2. On the **Workbench Configuration** tab, select the **Enable default Session checking** check box.

The next time you open a QALoad Script Development Workbench middleware session, you will be prompted to make it your default session.

QALoad can't find Tuxedo environment variable

If QALoad cannot find the TUXDIR environment variable, the environment variable contains an invalid value, or QALoad cannot find the directory `\tuxedo\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, QALoad may not be able to convert, compile, or play back your script at the appropriate point in your test.

Winsock running out of socket resources

You may encounter a problem running out of socket resources on NT or Solaris when there are large numbers of short-lived connections.

When TCP/IP connections are shutdown, they go into a TIME_WAIT state waiting for the specified interval to expire. While in that state the connection is looking for any stray packets that may have been sent to this connection and remain unacknowledged.

If this process was skipped, it would be possible for a new connection to be opened using the same address and port as the previous connection and to incorrectly receive data that was intended for the previous connection. When QALoad is generating many short-lived connections, during a Winsock or WWW load test, the default setting for the timed wait delay may be so high that the driver machine will run out of socket resources as all closed sockets wait in the TIME_WAIT state.

To change the setting for the timed wait delay:

Windows NT

Set the registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay
```

to a lower value. It can be set to anything between 30 and 300. Compuware suggests using the lowest possible value (30).

Solaris 2.6

Using "nnd" set the "tcp_close_wait_interval" to 30 seconds:

```
nnd -get /dev/tcp tcp_close_wait_interval
nnd -set /dev/tcp tcp_close_wait_interval 30000
```

Solaris 2.7

Use "nnd" as shown previously for Solaris 2.6, but substitute the "tcp_time_wait_interval"

SAP script validation fails

If your SAP script fails during validation, you may need to disable automatic proxy configuration in Internet Explorer.

To disable automatic proxy configuration:

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

If disabling the automatic proxy configuration does not solve the problem, consider increasing the script execution timeout value to 100 seconds or to the length of the capture file (in seconds), whichever is greater.

To increase the timeout value:

QALoad 5.02

1. With an SAP session open in the Script Development Workbench, click **Options>Workbench**.
2. On the **Script Validation** tab, type the new value in the **Wait up to** field.
3. Click **OK**.

Linking errors during validation or compilation of SAP scripts

When you re-record SAP 4.x scripts for SAP 6.20/SAP 6.40, you must click the Build SAP Libraries button on the [SAP Convert Options dialog box](#). This button generates new libraries based on the version of SAP that is currently installed. If you have upgraded to a newer version of SAP and do not update the libraries, you may experience various linking errors during validation or compilation.

Performance issues with SAP or Citrix scripts

If you experience performance issues with SAP or Citrix scripts, increase your system paging file size to a fixed size of at least four times the amount of RAM on the machine.

Conductor

Overview of the QALoad Conductor

You use the QALoad Conductor to configure, run, and monitor a load test that utilizes the scripts you created in the Script Development Workbench. The Conductor controls the QALoad Players and manages tests while they're running.

Before running a test, you must set up a test by recording descriptive information about the test, setting general test options, configuring Player workstations, assigning compiled test scripts to Players, and setting up monitoring options. Then, save the test setup in a file called a session ID. Once you've configured and saved a test session ID, you can reuse it without needing to re-enter any of your test information.

While a test is running, the Conductor interface changes to a tri-pane view called the Runtime window that facilitates monitoring of individual machines and Players, and displays real-time test results. You can view default graphs of performance data that are created for you by the Conductor and create custom graphs based on the data being collected during the test. Your custom graph layouts can be saved in the session ID file and reused in future tests.

About the Conductor

Overview of the QALoad Conductor

You use the QALoad Conductor to configure, run, and monitor a load test that utilizes the scripts you created in the Script Development Workbench. The Conductor controls the QALoad Players and manages tests while they're running.

Before running a test, you must set up a test by recording descriptive information about the test, setting general test options, configuring Player workstations, assigning compiled test scripts to Players, and setting up monitoring options. Then, save the test setup in a file called a session ID. Once you've configured and saved a test session ID, you can reuse it without needing to re-enter any of your test information.

While a test is running, the Conductor interface changes to a tri-pane view called the Runtime window that facilitates monitoring of individual machines and Players, and displays real-time test results. You can view default graphs of performance data that are created for you by the Conductor and create custom graphs based on the data being collected during the test. Your custom graph layouts can be saved in the session ID file and reused in future tests.

Taking a look at the Conductor

Taking a look at the Conductor

The Conductor's interface is dynamic — it changes depending on where you are in the testing process: setting up a test, or running a test. Both interfaces are described below.

Test setup

The Conductor's Main Window is divided into tabs on which you enter information about your test and set up the machines and scripts for the test. For more information about the test setup interface, see [Test setup interface](#).

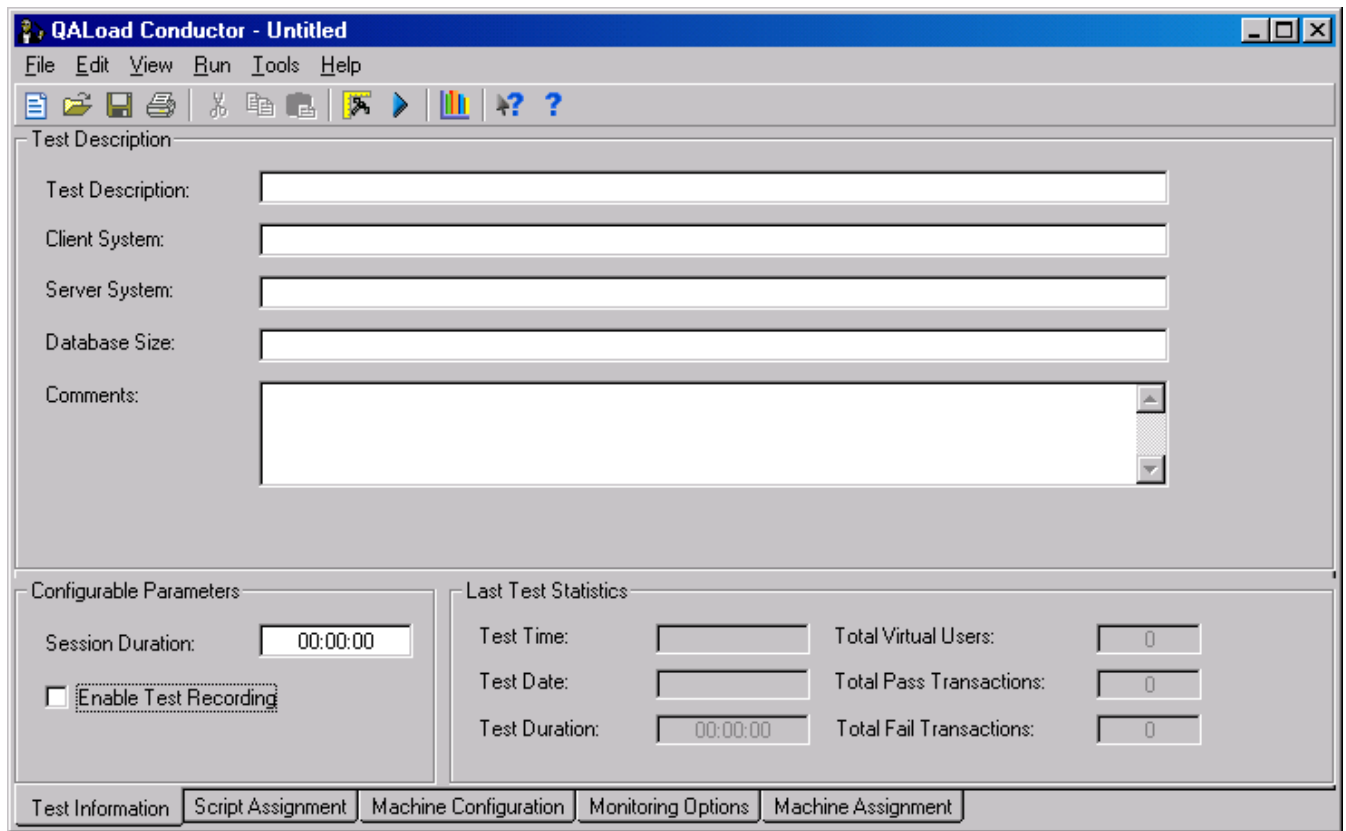
Running a test

While a test is running, the Main Window changes into the Runtime Window, which displays vital information about your running test and provides several controls to alter your test on-the-fly. For more information about the Runtime Window and how to use it while running a test, see [Running a test](#).

[Test setup](#)

Test setup interface

The Conductor's Main Window is divided into tabs on which you enter information about your test and set up the machines and scripts for the test. In addition, the Conductor's toolbar provides access to standard Windows functionality such as Print and Copy, as well as quick access to Conductor setup options and to QALoad Analyze. Use the tabs on the Main Window to set up your test. For detailed information about the fields on these tabs, click one of the links that follow or press F1 from any Conductor tab.



Field descriptions for each tab:

- [Test Information](#)
- [Script Assignment](#)
- [Machine Configuration](#)
- [Monitoring Options](#)
- [Machine Assignment](#)

Test Information

Use this tab to enter descriptive information about the test, view statistics from the previous test, and set a maximum amount of time for the current test to run. All descriptive information about the test will be incorporated into your test's timing file, and can be viewed in Analyze with your test statistics after the test has finished.

Test Description: Enter a description of the test, for example, its purpose. This field is written to your test's timing file to describe the test. This field is optional.

Client System: Enter a description of the client workstations. This field is optional.

Server System: Enter a description of the server(s) under test. This field is optional.

Database Size: Enter a description of the database used in this test. This field is optional.

Comments: Enter any general comments about the test in this field. Note that this field is overwritten at the end of a test if the Post Test Remarks check box on the Options dialog box, General tab is enabled. In that case, notes you type in the Post Test Remarks dialog will be saved to the test's Summary Report.

Session Duration: Enter a value to specify the maximum amount of time for the test to run. Entering zero (the default) will execute the test indefinitely. You can determine whether to stop the test immediately when the duration expires or to allow the virtual users to complete the running transactions by using the option Gracefully exit users when Session Duration expires on the [Options dialog box](#), General tab.

Enable Test Recording: Select this check box to [record load tests](#) for later review.

Test Time: This field displays the time the previous test started.

Test Date: This field displays the date the previous test started.

Test Duration: This field displays the length of time it took for the previous test to run.

Total Virtual Users: This field displays the number of virtual users assigned to the previous test.

Total Pass Transactions: This field displays the number of transactions that ran successfully in the previous test.

Total Fail Transactions: This field displays the number of transactions that failed in the previous test.

Script Assignment

Use this tab to set up any scripts that have previously been recorded and compiled. Any scripts you add here will be included in your load test, and one virtual user will be automatically assigned to your script on the Machine Assignment tab. After setting up your scripts here, you must assign additional virtual users to your script from the Machine Assignment tab.

Script: Select a script from the drop-down list, or click the New button to add a script.

New button: Click this button to browse for a script to add to the test.

Delete button: Click this button to remove the selected script from the load test.

Delete All: Click this button to delete all scripts from this test.

Type: Lists the middleware type of the script. Click the browse (...) button to set [custom middleware options](#).

Transactions: Type the number of transactions that each virtual user running the designated script should run. Once a workstation reaches the maximum number of transactions, the script continues execution with the line following the End_Transaction command rather than returning to the top of the transaction loop. If you enter 0, the Conductor will execute the script indefinitely.

Debug Options: Click the browse button to access the [Debug Options](#) dialog box, where you can configure debug settings for the script.

Error Handling: Choose how to respond when a non-critical error occurs during execution of the transaction. During large load tests, errors can sometimes indicate that the test is straining the limits of the hardware/software in the testing environment. Options are:

- ! Continue Transaction — If an error occurs while a transaction is being executed, the Player should continue executing the transaction as if the error had not occurred. Select this option when errors are not critical to the performance of the load test and can be safely ignored.


- ! **Abort Transaction** — If an error occurs while a transaction is being executed, the Player should abort the current transaction and the virtual user who encountered the error should exit the test. Use this option when errors will make the virtual user invalid for executing more transactions.
- ! **Restart Transaction (WWW, SAPGUI, and Citrix scripts only)** — If an error occurs while a transaction is being executed, the Player should abort the current transaction entirely and restart a new transaction from the beginning. Note that the transaction count will increase for each transaction that is restarted.

QALoad uses two commands — [DO_SetTransactionStart\(\)](#) and [DO_SetTransactionClean up\(\)](#) — to control error handling. These commands are inserted into your script by the conversion process.

Sleep Factor %: QALoad records the actual delays between requests and inserts the [DO_SLEEP](#) command in the script to mimic those delays when the script is played back in a test. You can maintain the exact length of the recorded delays at playback, or shorten them by entering a smaller percentage of the originally recorded delay to play back. For example, if you recorded a delay of 10 seconds then [DO_SLEEP \(10\)](#); is written to your script. Then, if a Sleep Factor of 50% is specified here, the Player will sleep for 5 seconds at that statement when the test is executed.

Valid values for Sleep Factor % are 0-100% and also Random, which will cause the Player to sleep for a randomly selected duration between 0 and the number of seconds specified in the [DO_SLEEP\(\)](#) statements.

When a load test is executed with a Sleep Factor of 100% the script executes at exactly the same speed at which it was recorded; therefore, you can simulate the performance of faster users by specifying a lower Sleep Factor % value.

 **Hint:** Enter a value of zero during unit testing to eliminate the actual sleeps from the script. After you unit test the script, you can restore the original recorded delays by changing the Sleep Factor to a higher percentage.

Service Level Threshold: Enter a response time to use as the threshold for comparing other response times. When you run a test, a line representing the Service Level Threshold will appear in the runtime window. As the test progresses, you can compare incoming response time data to the Service Level Threshold. This is a preliminary way of analyzing test results without waiting to open the timing file at the end of the test, and can be used as an indicator to determine if/when Dial-Up users should be added to the test.

Pacing: Enter a value in this field for the rate of pacing. Pacing is the time interval between the start of a transaction and the beginning of the next transaction on each workstation running the script. For example: if a transaction is designed to duplicate the process of someone handling incoming telephone calls and those calls arrive at a rate of 40 per hour/per person, set the pacing rate at 90 seconds.

The default pacing value is one second. This allows the Conductor to throttle runaway virtual users.

QALoad randomly schedules transactions so that each transaction executes on an average according to this predetermined rate. When a transaction completes faster than its pacing rate, QALoad delays the execution of the next transaction for that workstation so that proper pacing is met. Since we do not normally time events according to this predetermined rate, QALoad randomly accelerates or delays the pacing on a workstation-by-workstation basis. However, on the average, QALoad provides pacing according to the value that you assign.

Timing Options: Click the browse button to open the [Timing Options](#) dialog box, where you can choose to include QALoad's automatic timings in your test results or determine how much timing data to collect.

External Data: Click the browse button to open the [External Data](#) dialog box where you can select any central datapool files, local datapool files, or additional external support files necessary for your test.

Machine Configuration

Use the Machine Configuration tab to configure the various machines and agents that will participate in a load test. You can configure Player Machines, Server Analysis Agents, Remote Monitor Agents, and Application Expert information from a single screen.

You should use this tab to update Player or Agent information whenever a Player or Agent is added to the test network, removed from the test network, or the network address of a Player or Agent has changed.

The fields on this tab vary according to whether you choose the Player Machines, Server Analysis Agents, Remote Monitor Agents, or Application Expert option. Fields for Players and each agent type are described below.

Player Agents

Player machines execute the virtual users that will perform the transactions recorded in your test scripts. If no Player machines are listed, click the Discover button to retrieve information from Player machines on the local network, or you can [add Player machines manually](#).

Discover, add, delete, or modify Player machines using the following fields:

Agent Machine: Lists the hostname or IP address of the Player machine. You can edit the value, but it must be a valid hostname or dotted IP address so the Conductor can communicate with the machine during the test. Select the check box next to the Player machine to add it to a test, or clear the check box to disable the Player. Disabled Player Machines will not be available to run virtual users during the load test. Compuware recommends you verify the hostname/IP address before attempting to run a test by clicking Request for the selected Player machine.

Operating System: Lists the operating system of the Player machine. This information is retrieved from the Player machine when you use the Request or Discover options. If you have manually added a Player machine instead, this column will list Unknown for that machine until the machine has been successfully requested using Request or Discover. This information can be helpful in determining how many virtual users to allocate to a machine, or to help you determine if service packs should be applied to test machines.

Max Threads: Lists the maximum number of thread-based virtual users the Player machine is configured to execute. This value is automatically determined based on the settings in the Machine Defaults area on the Options dialog box. You can overwrite the default value for the selected Player machine by changing the amount in this field.

Max Processes: Lists the maximum number of process-based virtual users the Player machine is configured to execute. This value is automatically determined based on the settings in the Machine Defaults area on the Options dialog box. You can overwrite the default value for the selected Player machine by changing the amount in this field.

Performance Data: Click to access the [Performance Data Options dialog box](#), where you enable performance data collection.

Discover Machines: Click Discover Machines for QALoad Conductor to query the network for available Player workstations and display the results on this tab.

Request: Click Request for the Conductor to send a request to the selected Player to verify that it is up-and-running.

Request All: Click to request all assigned Players to ensure that they are up-and-running.

New: Click to access the [New Entry dialog box](#) where you can configure a Player workstation to add to the test.

Delete: Click to delete the selected entry.

Delete All: Click to delete the entire list.

Server Analysis Agents

Enable Data Capture: Select to capture data from the Agents at runtime.

Add Templates to Selected Agent: Click to navigate to a pre-defined template of counters to assign to this Agent machine.

Discover All Agent Counters: Click to query the network for workstations with Server Analysis Agents installed, and to load information about the counters available from each. The results will be listed in the tree-view.

Discover Machines: Click Discover Machines for QALoad Conductor to query the network for available Server Analysis Agent workstations and display the results on this tab.

New: Click New to access the [New Entry dialog box](#) where you can configure a Server Analysis Agent workstation to add to the test.

Delete: Click this button to remove the selected Agent workstation from the tree-view.

Delete All: Click to remove all Server Analysis Agents from the test.

Remote Monitor Machines

Enable Data Capture: Select to capture data from the Agents at runtime.

New: Click New to access the Remote Machine Configuration dialog box, from which you can configure a Remote Monitor Agent workstation to add to the test.

Delete: Click this button to remove the selected Agent workstation from the tree-view.


Delete All: Click to remove all Remote Monitor Agents from the test.

Application Expert

 **Note:** Before you can collect network performance data from Application Expert, it must be properly installed and configured. For details, see [Application Expert overview](#).

Machines to Add: Select from the list of machines that have Player agents installed and use the arrow buttons to move the machine name into the IP Pairs area as Address 1.

IP Pairs: In the Address 1 and Address 2 columns, add the machine names between which you want to monitor traffic. Address 2 should be the IP or machine name of the first tier of the application being tested, or you can simply type *any* to capture all traffic for Address 1. You can specify machines with URLs or IP addresses. Select the Include checkbox to specifically monitor those machines, or clear the check box to turn monitoring off for those machines.

 **Note:** If the network environment is running over a switch, then Address 1 must be the IP address of the Conductor machine and the virtual user to capture must execute from that machine. If the network environment is running over a hub (shared network), then Address 1 can be any Player machine available for testing. For more information, see [Configuring a test to use Application Expert](#).

Host Name: This field automatically lists the name of the machine where the Conductor is installed. Do not change this.

Username: Type a valid user name for the machine listed in the Host Name field.

Password: Type the password corresponding to the Username.

NIC Name: From the list, select the NIC (network interface card) the machine listed in the Host Name field should use.

Monitoring Options

Use the Monitoring Options tab to specify options for integrating ServerVantage into your load test. QALoad assumes that the appropriate ServerVantage software is installed, configured, and running prior to starting a load test.

Enable ServerVantage Integration: Select if you are integrating with ServerVantage.

Control Server Hostname: Type the hostname of the machine where the ServerVantage server is located.

Username: Type a valid user name to access the ServerVantage server.

Password: Type the password corresponding to the user name above.

Override Default Database: Select to provide the ServerVantage database name. When this option is not selected, QALoad uses the default ServerVantage database name. If you provided a different name during the installation of ServerVantage, select this option and type the name in the Database Name field.

Database Name: If you selected the Override Default Database option, type the name of your ServerVantage database.

Vantage Agent Configuration: Type the hostname of a machine(s) where a ServerVantage Agent is installed, and click Add to add it to your load test.

Agents to Monitor: Lists the machine names of each ServerVantage agent being monitored for this test.

Remove: To remove an agent machine from your load test, select it in the Agents to Monitor area and then click Remove.

Machine Assignment

Use the Machine Assignment tab to assign scripts to specific Player workstations. You can use the Edit menu's Copy and Paste commands to copy and paste machine entries (rows) as needed.

Script: This field displays the script name. To add a script, click New.

Middleware: Displays the middleware type the script was created for.

Starting VUs: Type number of virtual users to begin the test.

VU Increment: Type the number of virtual users to be added, at intervals, after the test begins.

Time Interval: Displays the time interval at which incremental virtual users will be added to a test. Change the time interval by typing a new value.

Ending VUs: Displays the number of virtual users assigned to run until the end of the test.

Machine: From the drop-down list, select Player machines to assign each script to run on. For Application Expert and ApplicationVantage integrations, the Player and Conductor machines must be on the same LAN. For more information about integrating with Application Expert or ApplicationVantage, see [Application Expert/Application Vantage overview](#).

Mode: Select the test mode for each Player machine: thread-based, process-based.

New button: Click to access the Select Script dialog box where you can select a script to add to the test.

Delete: Click to delete the selected script.

Delete All: Click to delete all scripts.

Auto Configure: Click this button to have QALoad automatically assign scripts to virtual users.

Run: Click to start a test run. This button is only enabled if your test is completely set up.

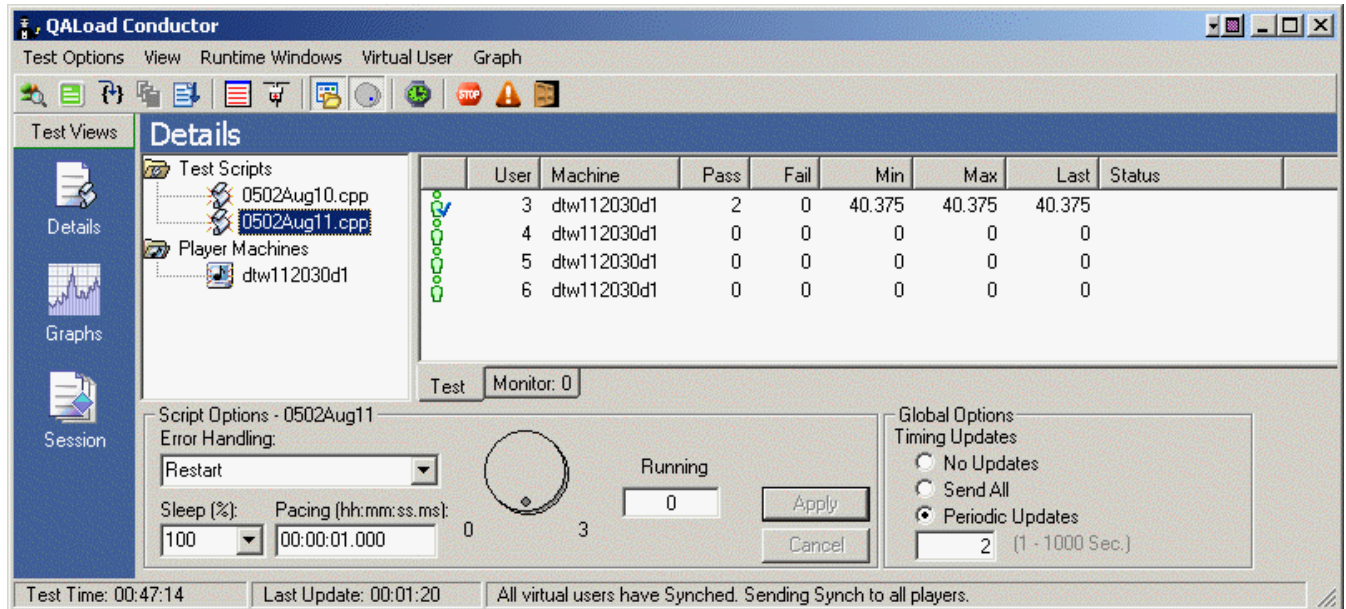
Running a test

Runtime window interface

When you start a test, the Conductor's interface changes to an interactive test control station called the Runtime Window. The Runtime Window displays information about the scripts, machines, and virtual users that are executing the load test. The test data is divided into three views – Details, Graphs, and Session – that are accessed from the Test Views workspace on the left side of the interface.

From the Runtime Window, you can observe the progress of individual scripts and Player machines, view real-time graphs, and start or suspend scripts and Players from a running test to better simulate the

unpredictability of real users. This window has three unique areas. Click on a pane in the following graphic for a brief description of that pane. Or, click on the links below the graphic for detailed information.



Details view

The Details view of the Data window displays all your test data in real-time in a series of interactive tabs. By clicking on icons representing scripts, virtual users, and workstations, you can view different types of data. By default, each test displays test details in the right pane. You can also choose to view the script a single virtual user is running, the Web page a WWW script is utilizing, or the RIP file generated by a failed virtual user.

Test details

Test details display automatically, and describe the object you select in the tree view. You can view details for all test scripts, individual test scripts, all player machines, and individual player machines.

See [Test details](#) for more information about the Tree items that can be displayed in the data window.

Runtime tabs

The following runtime tabs can be displayed for a running script. These tabs can be enabled from the Virtual User menu.

- [Debug](#)
- [Monitor \(Web User\)](#)
- [RIP File \(WWW\)](#)

Runtime Control Panel

The Runtime Control Panel is a dockable control station that enables you to change virtual user options and data transfer options while the load test is running. For more information, see [Runtime Control Panel](#).

Graphs view

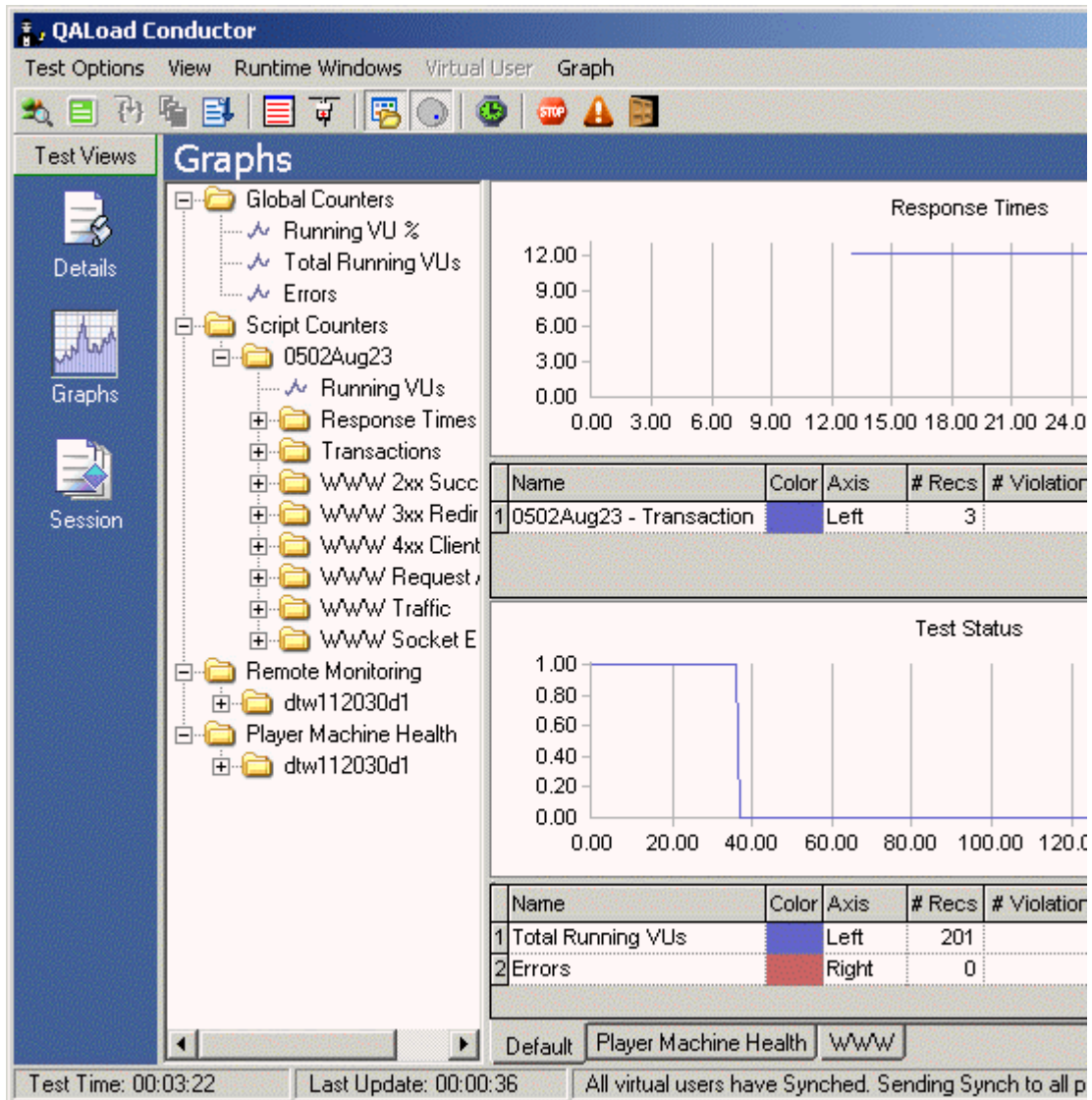
The Graphs view displays graphs of data collected during the test. By default, the Graphs view displays graphs for response times, test status, and [player machine health](#).

Other graphs, such as user-defined checkpoints and Remote Monitoring counters, can also be plotted in the right pane of the Graphs view if they were enabled for the session.

To display graphs:


1. Right-click on a counter or other data type in the tree view that you want to plot in a graph.
2. Choose **Add Graph** or **Add Plot To**.

You can also modify a graph's appearance by right-clicking on the graph and choosing one of the formatting options, such as colors and axes properties. To increase the visibility of a plot when you have multiple plots on a graph, click on a plot (or that plot's number in the legend) to highlight it.



Session view

The Session view provides summary information about the test session that is currently running. The Session view can be printed as a report by right-clicking and choosing Print from the shortcut menu.

 **Note:** The Session view below has been cropped to better fit this help topic, while still representing what a real Session view might look like.

Click on the sections in the following graphic for more information about the Session view.

Current Summary

Running Scripts

Script	Response Time	Total VUs	Running VUs	Pass Transactions	Fail Transactions	Throughput
0502Aug10	773.83	3	0	5	0	0.00/s
0502Aug11	40.38	4	0	2	0	0.03/s

Session Summary

Test Information

Session ID Name	0811session.id
Conductor Build	05.02.00 Build 090
Session Duration	00:00:00
Total Scripts	2
Total Players	1
Total Virtual Users	7
Total Running Virtual Users	0

Script Information

0502Aug10

Path	C:\Program Files\Compuware\QALoad\Middlewares\WWW\Scripts\0502Aug10.cpp
Middeware Type	WWW
Transactions	5
Automatic Timings	Enabled
Include Sleep Times	False
Checkpoint Thinning	Disabled
Counter Data Collection	Store in Timing File and Display in Conductor
Counter Thinning	By Script Every 1 second(s)
Sleep Factor	100%
Transaction Peding	00:00:01.000
Service Level Threshold	00:00:00
Error Handling	Restart Transaction
Central Databool	None

0502Aug11

Path	C:\Program Files\Compuware\QALoad\Middlewares\WWW\Scripts\0502Aug11.cpp
Middeware Type	WWW
Transactions	2
Automatic Timings	Enabled
Include Sleep Times	False
Checkpoint Thinning	Disabled
Counter Data Collection	Store in Timing File and Display in Conductor
Counter Thinning	By Script Every 1 second(s)
Sleep Factor	100%
Transaction Peding	00:00:01.000
Service Level Threshold	00:00:00
Error Handling	Restart Transaction
Central Databool	None

Machine Information

Machines In Test

Hostname	OS	RAM	Processor
dtw112030d1	Windows 2000 Workstation Service Pack 3	1023 MB	Intel Pentium 4

Machine Assignments

Script	Start VUs	VU Increment	Interval	End VUs	Machine	Node
0502Aug10	1	0	00:00:00	3	dtw112030d1	Thread
0502Aug11	1	0	00:00:00	4	dtw112030d1	Thread

QALoad Conductor menus and toolbar buttons

The Conductor's menus and toolbar buttons are dynamic; their content depends on whether you are preparing a test setup or running a test.

Test setup

The Conductor Configuration and Setup Menus allow you to configure the Conductor and your specific test. Click a menu or toolbar name below for details:

- File
- Edit
- View
- Run
- Tools
- Help
- Configuration and Setup toolbar buttons

Running a test

The Conductor's Runtime menus and toolbar allow you to control your running test and the data that is displayed at test time. Click a menu name below for details:

[Test Options](#)

[View](#)

[Runtime Windows](#)

[Virtual User](#)

[Graph](#)

[Runtime toolbar buttons](#)

Server and performance monitoring

QALoad integrates several mechanisms for merging load test response time data with server utilization data and performance metrics. Select the method that best suits your needs, or for which you are licensed (if applicable). Most methods produce data that is included in your load test timing results and processed in QALoad Analyze. The only exception is Application Vantage. Data captured from Application Vantage can be opened in Application Vantage or Application Expert, but not in QALoad.


This section briefly describes each method, and provides links to more detailed information about setting up a test that includes the appropriate method.

- ! [Remote Monitoring](#) — allows you to monitor server utilization statistics from a remote machine without installing any software on the remote machine.
- ! [Server Analysis Agents](#) — must be installed on each applicable machine.
- ! [ServerVantage](#) — integrates with your existing ServerVantage installation. You must be licensed for and have installed and configured the appropriate product in order to integrate with QALoad .
- ! [Application Expert/ApplicationVantage](#) — collects test data that you can open in both Application Expert and Application Vantage.

Monitoring CPU usage

To help you monitor the impact of running a load test on a server, QALoad can collect data from selected Players about CPU usage during a load test. The statistics collected during the test are merged into the test's timing file so you can view them in Analyze after the test.

When the Top Process Monitoring counter is enabled for a Server Analysis Agent machine, information will be collected periodically during the load test about which processes are using the most CPU. The Conductor will collect information for up to ten processes during the test, but sometimes less if some processes are not affecting CPU usage. Counter data will be written to your test's timing file, which you can open in Analyze after your test.

 **Note:** During a load test, if the CPU idle time of your machine falls below 25%, check the individual processes on your machine. If the Players and virtual users are utilizing most of the active CPU time, you should use additional Player machines and fewer virtual users per Player to conduct your load test.

QALoad can compile statistics from all processes currently running on the following platforms: Windows 2000 and XP, HP-UX, Linux, and Solaris.

To collect Top Processes data:

1. With your test session ID file open, click the **Machine Configuration** tab.
2. Click the **Server Analysis Agents** option that is directly beneath the toolbar.

3. If necessary, click **Discover Machines** so the Conductor can locate the available Server Analysis Agent machines on your network.
4. To enable Top Process Monitoring on any Server Analysis Agent machine, click on the machine name to expand the tree-view of available counters for that machine. Then, select the **Top Process Monitoring counter**.
5. To save your changes to your test session ID file, select **File>Save** from the Conductor menu.

When your test is finished, the Top Processes data collected will be included in your test's timing file which you can open in QALoad Analyze.

Dial-up virtual users

QALoad's dial-up/dial-down feature allows you to dynamically add or reduce virtual users to your test at the script or Player level while your test is running, so you can adjust your running test according to test behavior on-the-fly, rather than stopping to re-configure playback criteria.

To use the dial-up/dial-down feature, you:

- ! must be licensed for at least the number of virtual users requested
- ! must configure a ramp-up session before running the test

Setting up a test

Overview of test setup

To set up a load test, you will set options related to general Conductor behavior as well as information about your specific test environment. Before you can successfully set up a load test, you must have recorded and compiled one or more test scripts. For information about recording a test script, see [Developing scripts](#) in the Getting Started section.

Determining general Conductor behavior

General Conductor options you set will be applicable for all tests run until you change them. Conductor options are related to the following:

- ! Viewing options for real-time results
- ! Global Player options
- ! Player machine performance data
- ! Options for runtime reporting
- ! And more...

All of the above information, and more, can be configured from the Conductor's [Options dialog box](#).

Setting up a specific test session

To prepare the Conductor for a specific test, you will save information and parameters specific to that test into a reusable session ID file (.id). You will need to enter the following types of information to set up a test's session ID file:

- ! General information about the test such as a description, the size of the database, the length of the test, and any notes or comments
- ! Information about the test script(s) included in the test, including script name, middleware/protocol type, pacing, whether to include external data, and so on
- ! Information about the workstations where the QALoad Players reside, including which script is assigned to each workstation, how many virtual users are assigned to each workstation, the machine name, and so on
- ! (Optional) configuration for server monitoring
- ! (Optional) integration with other Compuware products.

All of the above information can be entered and saved from the Conductor's main window, the [Test Information Window](#).

Anticipating error conditions

You know before beginning a load test that errors are a possibility, but you may not always want them to stop you cold.

QALoad helps you anticipate error conditions and determine, before running your test, how your Players will react to non-fatal errors. By setting one option, you can instruct a Player to continue as if no error was encountered, stop running immediately, or restart at the beginning of the transaction.

To set the error handling option, see the help topic for the [Script Assignment tab](#).

Configuring the Conductor

There are several settings for the Conductor that you should review before beginning your load test.

To set Conductor options that are not specific to one test:

1. From the main menu, choose **Tools>Options**.
2. On the Options dialog box, set options related to post-test activity, warnings and prompts, runtime grids, timing settings, interface refresh intervals, Conductor/Player communications, monitoring intervals, and more.
3. When you are finished, click **OK** to save your changes. Any options you set will apply to all tests until you change them.

For detailed descriptions of the options that are available, see [Options dialog box](#).

Managing large amounts of test data

Your load test will probably include a large number of checkpoints and virtual users in order to adequately test your system. When your test is running and your Conductor is collecting timing information from your Player machines, the sheer amount of data can take up more of your resources than you'd like to expend. Use QALoad's Timing Data Thinning option to thin the amount of timing data being transferred back to the Conductor during the test so that your test can run longer without stressing your resources.

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

For more details about the Timing Options dialog box, see [Timing Options](#).

Setting up a test session

You can enter all the information necessary for your session ID file right from the Conductor's main window, the Test Information Window.

i Hint: The following procedures guide you through setting up a reusable test session ID using the Conductor's main window, the Test Information window. Follow each step in turn to configure your test, or revisit this help topic later to make changes to any specific part of the test setup.

Step 1: Enter descriptive information about the test

On the Test Information tab:

1. (Optional) Type descriptive information about the test in the **Test Description**, **Client System**, **Server System**, **Database Size**, and **Comments** fields.
2. In the **Session Duration** field, type a time limit to specify the maximum duration for the test to run. Enter zero if you do not want to specify a maximum duration.

i Hint: For details about any field on the Test Information tab, see [Test Information](#).

Step 2: Assign compiled scripts to the test

On the Script Assignment tab:

1. Click **New** to open the Select Script dialog box. The Select Script dialog box lists the scripts available for your transaction type. If it does not, select your transaction type (middleware environment) from the **Scripts of Type** list.

i Hint: To open the Select Script dialog box from the **Script** column, click in the **Script** column to enable the **Browse (...)** button. Then, click **Browse**.

The Select Script dialog box lists a status for each script that indicates whether the script is compiled. If it is not, you must compile the script before attempting to use it in a test.

2. Select a script from the list and click **Select** to return to the **Script Assignment** tab.
3. Continue selecting scripts until all scripts you wish to use in this test are listed.

i Hint: For details about any field on the **Script Assignment** tab, see [Script Assignment](#).

Step 3: Set test options for each script

For each assigned script on the Script Assignment tab:

1. In the **Transactions** column, type the number of transactions that each virtual user running this script should run. Once a workstation executes the number of transactions that you specify, script execution continues with the line following the End_Transaction command rather than jumping to the beginning of the transaction loop
2. Click in the **Debug Options** column to enable the **Browse** button. Click **Browse** to open the Debug Options dialog box, and then set any options for **Debug Trace** and **Logfile generation**. For a description of the Debug Options dialog box, see [Debug Options](#).
3. In the **Error Handling** column, select the option that indicates how the Player running this script should behave when encountering non-fatal errors: **Abort the transaction**, **Continue the transaction**, or **Restart the transaction**.
4. Enter a value in the **Sleep Factor** column to specify the percentage of any originally recorded delay to preserve in the script (for example, a value of 80 means preserve 80% of the original delay).
5. In the **Service Level Threshold** column, type a maximum duration for this script. At runtime, the QALoad Conductor will display a runtime graph comparing the **Service Level Threshold** with the actual duration.
6. In the **Pacing** column, type a value, in seconds, for pacing.
7. Click in the **Timing Options** column to enable the **Browse** button. Then, click **Browse** to access the [Timing Options](#) dialog box and set options related to checkpoints and data thinning.
8. (Optional) Click in the **External Data** column to enable the **Browse** button. Then, click **Browse** to open the [External Data](#) dialog box and associate any necessary external files with your selected script.

i Hint: For details about any field on the **Script Assignment** tab, see [Script Assignment](#).


Step 4: Set up Player machines

On the Machine Configuration tab:

1. Select the **Player Agents** option.
2. Click **Discover Machines** to query your network for QALoad Player workstations. All workstations with QALoad Players installed will be listed. If Player machines are discovered to have previous versions of QALoad installed, an error message will inform you which machines need to be updated.
3. Check the availability of all the Player machines on your network by clicking **Request All**, or by selecting individual machines and clicking **Request**. The QALoad Conductor will request each selected Player machine to ensure it is available.

If a Player machine is available, system information for that machine will appear in the Properties dialog box for that Player machine (double-click on the Player machine listing to access its Properties dialog box). If the Player machine is not available, you will receive a message that the Player is not responding.


4. (Optional) Use **New** to manually add a Player machine, or **Delete** or **Delete All** to remove machines. To save the current machine setup for re-use, create a new configuration file (.cfg). [How?](#)

 **Hint:** For details about any field on the Machine Configuration tab, see [Machine Configuration](#).

Step 5: (Optional) Set up Server Analysis Agents

On the Machine Configuration tab:

1. Select the **Server Analysis Agents** option.
2. Click the **Enable Data Capture** check box to enable monitoring at test time.
3. Click **Discover Machines** to query your network for workstations with Server Analysis Agents installed. If machines are discovered to have previous versions of QALoad installed, an error message will inform you which machines need to be updated.
4. Determine which counters are available:
 - All Server Analysis Agents — Click Discover All Agent Counters.
 - Specific Server Analysis Agents — Double-click on the machine name.
5. Select counters to monitor at test time. Expand the tree-view for a machine name to view the available counters, and then click on the check box next to a counter to select it.
6. (Optional) Use **New** to manually add a Server Analysis Agent, or **Delete** or **Delete All** to remove them. To save the current machine setup for re-use, create a new configuration file (.cfg). [How?](#)

 **Hint:** For details about any field on the Machine Configuration tab, see [Machine Configuration](#).

Step 6: (Optional) Set up Remote Monitor Machines

On the Machine Configuration tab:

1. Select the **Remote Monitor Machines** option.
2. Click the **Enable Data Capture** check box to enable monitoring at test time.
3. Click **Discover Machines** to query your network for available machines.
4. Determine which counters are available:
 - All Remote Monitor Agents — Click Discover All Agent Counters.
 - Specific Remote Monitor Agents — Double-click on the machine name.
5. Select counters to monitor at test time. Expand the tree-view for a machine name to view the available counters, and then click on the check box next to a counter to select it.
6. (Optional) Use **New** to manually add a Remote Monitor machine, or **Delete** or **Delete All** to remove them. To save the current machine setup for re-use, create a new configuration file (.cfg). [How?](#)


 **Hint:** For details about any field on the Machine Configuration tab, see [Machine Configuration](#).

Step 7: (Optional) Set up integration with Application Expert

On the Machine Configuration tab:


1. Select the **Application Expert** option.
2. Click the **Enable Data Capture** check box to enable monitoring at test time.
3. In the **Machines to Add** area, choose the Player machine that will be running the virtual user to be captured. To make your selection, highlight the machine name and click the arrow button to move the machine name into the **IP Pairs** area as Address 1.

Address 2 should be the IP or machine name of the first tier of the application being tested, or you can simply type *any* to capture all traffic for Address 1. Select the Include check box to monitor communications between those two machines during your load test.

 **Note:** If the network environment is running over a switch, then Address 1 must be the IP of the Conductor machine and the virtual user to capture must execute from that machine. If the network environment is running over a hub (shared network), then Address 1 can be any Player machine available for testing. For more information, see [Configuring a test to use Application Expert](#).

4. The **Login Area** section should only be modified if you have installed the ApplicationVantage Remote Agent and not the QALoad Vantage Agent. These fields will contain the network information for the workstation where your ApplicationVantage Remote Agent is installed.
 - a. **Host Name:** This field automatically lists the name of the machine where your Conductor is installed. Do not change this.
 - b. **Username:** Type the user name used during the installation of the ApplicationVantage Remote Agent. If using the QALoad Vantage Agent, type the username *Admin*.
 - c. **Password:** Type the password used during the installation of the ApplicationVantage Remote Agent. If using the QALoad Vantage Agent, type the password *Admin*.
 - d. **NIC Name:** From the drop-down list, select the NIC (network interface card) that is used by the machine in the Host Name field.
 - e. (Optional) To save the current machine setup for re-use, create a new configuration file (.cfg). [How?](#)

At test time, the Conductor will pass this information to your installation of Application Expert, which will then capture the communications between the specified pairs of machines and save that information to a file you can open in Application Expert after the test has finished. The file, with an .opt extension, will be saved to the \temp directory of the current user's profile directory.

 **Hint:** For details about any field on the Machine Configuration tab, see [Machine Configuration](#).

Step 8: (Optional) Set up integration with ServerVantage

On the Monitoring Options tab:

1. Click the **Enable ServerVantage Integration** check box.
2. In the **Control Server Database Host** field, type the hostname of the machine where the ServerVantage server is located.
3. In the **Username** field, type a valid user name to access the ServerVantage server.
4. In the **Password** field, type the password corresponding to the user name above.
5. Select the **Override Default Database** check box to provide the ServerVantage database name. When this option is not selected, QALoad uses the default ServerVantage database name. If you provided a different name during the installation of ServerVantage, select this option and type the name in the **Database Name** field.
6. In the **Vantage Agent Configuration** area, type the hostname of a machine(s) where a ServerVantage Agent is installed, and click the **Add** button to add it to your load test.

Step 9: Assign scripts to Player machines

On the Machine Assignment tab, the scripts you assigned to the test on the Script Assignment tab are listed in the Script column. Fill in the following columns:

 **Note:** Use **Auto Configure** to have QALoad automatically assign scripts to virtual users.

1. In the **Starting VUs** column, type the number of virtual users to initially launch the script on this machine when a test begins.
2. In the **VU Increment** column, type the number of virtual users that should be added, at intervals, if you want this machine to add incremental virtual users. You must also fill in the **Time Interval** and **Ending VUs** fields.
3. In the **Time Interval** column, type the time interval at which incremental virtual users should be added to a test. (For example, to add virtual users every 5 minutes, type 00:05:00). You must also fill in the **VU Increment** and **Ending VUs** field.
4. Type the number of virtual users assigned to run until the end of the test.
5. In the **Machine** column for each script, select a Player machine from the drop-down list to assign it to that script. If no Player machines are available in the drop-down list, click the **Machine Configuration** tab to set up a Player.
6. In the **Mode** column, select the test mode for each Player machine: **thread-based** or **process-based**.
7. (Optional) Use **New**, **Delete**, and **Delete All** to add or remove scripts from this test.

When all scripts have been successfully assigned to Player machines and the test is ready to run, Run on the Machine Assignment tab will become available and you can run a test.

Step 10: Save the test setup you just created as a reusable session ID file

Save the test setup

To save the current test setup to a reusable test file called a session ID, click File>Save to name and save it.

Save the machine configurations

To save the Player Agent, Server Analysis Agent, Remote Monitoring Agent, and ApplicationVantage integration configurations to a reusable file, called a configuration file (.cfg) see [Saving machine configurations](#).

Saving machine configurations

After configuring the machines to use for a load test, you can save the machine configuration information into a configuration file (.cfg) that can be reused in later tests, saving you significant time setting up later tests. A configuration file includes information about which machines on the network were used as Player Agent, Server Analysis Agent, Remote Monitoring, and ApplicationVantage machines. You can save multiple configurations under different names. By default, when first using QALoad, the Conductor uses a configuration file named `Default.cfg`. The Conductor will save any changes to your machine configurations to this file unless you save your configuration to a new file with a different name.

You can open or save .cfg files from the Machine Configuration tab. The .cfg field always displays the active configuration.

To create a new, empty .cfg file:

1. On the **Machine Configuration** tab, click on the drop-down .cfg field at the bottom of the tab.
2. Choose **<New>**.
3. On the Save As dialog box, specify a name for the new file and click **Save**.
4. Add the necessary Player and agent machines using the fields and buttons on the **Machine Configuration** tab. The machines you configure are saved automatically to the file you just created.


To rename the current .cfg file:

1. On the **Machine Configuration** tab, click on the drop-down .cfg field at the bottom of the tab.
2. Choose **<Save As>**.

3. On the Save As dialog box, specify a name for the new file and click **Save**.
4. Make any necessary changes to the configuration. Your changes are saved automatically to the file you just created.

To open a previously created .cfg file:

1. On the **Machine Configuration** tab, click on the drop-down .cfg field at the bottom of the tab.
2. Choose the .cfg file to open.

 **Note:** The .cfg file only stores information about Player machines and counter agent machines. It does not store information specific to a test, such as script names or settings. Test specific information is saved in the session ID file. A session ID file for a specific test will save the name of the .cfg file associated with that test, and open it automatically when the session ID file is opened. You can change the .cfg file at any time without needing be concerned about the session ID file.

Add a Player workstation to a test session ID

Follow these instructions to add a Player workstation to your pool of available Players in a test's session ID file.

To add a Player workstation to a test session ID:

1. On the Conductor's Test Information window, click the Machine Configuration tab.
2. Click the **New** button to open the **New Entry dialog box**.
3. Type the host name and port number of the new Player in the **Agent Machine Host Name** and **Agent Port** fields. If you do not know a Player's host name, check the Player's window at startup. It displays the workstation's Player name.
4. Click the **Request** button to ensure the machine is available. If the Player is available, the number of thread- and process-based virtual users it supports are listed in the appropriate columns, and the button text changes to **OK**.

If the Player does not respond, a message box appears indicating that the Player is not responding. If the Player is not responding, one of the following scenarios is likely:

- The host name and/or port number you entered may not be correct. Check your parameters and network connections, then try to send another request.
- The Player is not running. Start the Player and then try to send another request.

Adding sessions to a batch test

Before a session is added, the following conditions must be true:

- ! The session must include a defined number of transactions. Sessions of unlimited transactions cannot be used in a batch test.
- ! All scripts must exist prior to starting the batch test. This means that the files referenced in the selected session ID files are present in the script directory.

A session can be placed in a batch multiple times. This feature might be used to re-run a test or to perform housekeeping chores, such as logging users in or out of a host or database.

To add a session:

1. From the **Run** menu, choose **Batch Test**.
2. In the **Session Files (.id)** box, highlight the session you want to add, and click the **Add** button.

If you want to run a previously defined batch, click the Load button to navigate to the directory where the batch file (.run) resides. Select it, and click OK.

The session is added to the Batch List on the right side of the dialog box.

Adding a script to a test

To add a script to a test session:

1. From the Test Information Window, click the Script Assignment tab.
2. Click the **Browse** button in the Script column to open the Select Script dialog box.
3. From the **Scripts of Type** box, select your script type.
4. From the list of available scripts that appears, highlight a script name and click the **Select** button. You are returned to the Test Information Screen.
5. In the Transactions column, specify the maximum number of transactions that you want each virtual user running this script to run. Once a workstation executes the number of transactions that you specify, script execution continues with the line following the End_Transaction command rather than jumping to the beginning of the transaction loop.
6. Click the **browse [...]** button in the Debug Options column to access the [Debug Options](#) dialog box, where you can specify virtual users for debug trace and logfile monitoring.
7. In the Error Handling column, select what to do when the script encounters an error: Continue Transaction (as if no error had been encountered), Abort Transaction, or Restart Transaction (WWW, SAPGUI, and Citrix scripts only).
8. Enter a value in the Sleep Factor % column to specify the percentage of any originally-recorded delay to preserve in the script (for example, a value of 80 means preserve 80% of the original delay). Valid values are 0-100, or Random. The default value is 100%.
9. In the Service Level Threshold column, enter a response time by which to compare incoming response times during a test. When you run a test, a line representing the Service Level Threshold will appear on the runtime graph. As the test progresses, you can compare incoming response time data to the Service Level Threshold.
10. Enter a value, in seconds, in the **Pacing** field. [Pacing](#) is the time interval between the start of a transaction and the start of the next transaction for each virtual user running a script.
11. In the Timing Options column, click the **browse [...]** button to access the Timing Options dialog box where you can configure how much timing data is collected. For details about the Timing Options dialog box, click [Timing Options](#).
12. Click the **browse [...]** button in the External Data column to access the External Data dialog box where you can select a datapool or other file to include with your test. For details about the External Data dialog box, see [External Data](#).
13. From the **File** menu, choose **Save** to save your changes to the current session ID file, or **Save As** to save them to a new session ID file.

Changing the number of virtual users

You can change the number of virtual users assigned to a script from the Test Information Window, Machine Assignment tab.

To change the number of virtual users:

1. Enter a new value in the Starting VUs column for the selected script.
2. If you have assigned incremental virtual users, change the values in the VU Increment and Ending Vus columns for the appropriate script to determine how many virtual users to add at the interval specified in the Time Interval column.

3. From the **File** menu, choose **Save** to save your changes to the current session ID file, or **Save As** to save them to a new session ID file.

Changing test options

To change test options:

1. To change any of your test options, make your changes on the Test Information, Script Assignment, Machine Configuration, Monitoring Options, or Machine Assignment tabs.
2. From the **File** menu, choose **Save** to save your changes to the current session ID file, or **Save As** to save them to a new session ID file.

Removing a script from a test

1. On the **Test Information Screen Script Assignment** tab, click on the selection box to the left of the script name to highlight the row.
2. Click **Delete** to remove the script from the test.
3. Select **File>Save** to save your changes to the current session ID file, or **File>Save As** to save them to a new session ID file.

Removing a session from a batch test

1. Select **Run>Batch Test**.
2. Highlight the session to remove in the Batch List and click **Remove**.

Setting Auto Abort

1. Select **Run>Batch Test**.
2. Select **Auto Abort After** and use the slider to set the number of seconds to wait before aborting a test.

Normally, each test runs for the duration set in its respective session ID file. An individual test run is complete when all the virtual users have exited. If the Auto Abort After check box is selected and all the virtual users in a test do not exit within the specified number of seconds, the Conductor automatically aborts the test.

Remove used datapool records after a test

You can remove used datapool records after a test by setting the Strip Datapool function before you run the test. Use this function when running a test where you have data in the external datapool which can only be used once by one virtual user at a time. (For example, when running transactions that have unique data constraints.) When activated, the Strip Datapool function will mark each piece of data in the datapool that is used during your test. When the test is over, the Strip Datapool function prompts you to remove the identified used data from the datapool. If you run the test again, only new data will be used for your subsequent test.

To use the Strip Datapool function:


1. With the current test's session ID file open in the Conductor, select the Script Assignment tab.
2. Click the **External Data** button. The External Data dialog box appears.

3. In the Central Datapool area, select the **Strip** check box. Click **OK**.
4. At the end of your test, a Strip Datapools prompt will appear asking if you wish to go to the Strip Datapools screen. Click **Yes**.
5. The Strip Data Pool dialog box appears. Click the **Strip** button.
6. When you are finished, click **Done**.

Setting automatic stripping of datapools between batch tests

1. Select **Run>Batch Test**.
2. Add sessions to the Batch List, or load the batch file you wish to run.
3. Select **Automatic Datapool Stripping Between Tests**.

Datapool records used during the test will be removed before starting the next test in the batch.

 **Note:** Only those datapools marked as strippable on the External Data dialog box, accessible from the **Test Information Screen Script Assignment** tab, will be removed.

Setting delays between tests

You can set a fixed delay or pause between tests by specifying a value in the Delay Between Tests field on the Batch Test dialog box. After each test is complete, the Conductor delays for the specified amount of time before starting the next test.

Validating a script

Before running a test, you should run your script in a simple test to ensure that it runs without errors. You can validate UNIX or Win32 scripts from the Conductor:

[UNIX scripts](#)

[Win32 scripts](#)

Debugging a script

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

[Watch a virtual user execute a script on a Player workstation while it is running](#)

To monitor selected virtual users at runtime, enable the Debug Trace option before you run your test.

To enable the Debug Trace option:

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

6. Run your test as usual. Each virtual user for which you enabled Debug Trace will display messages on its assigned Player workstation indicating which commands are being executed.

[Log details from selected virtual users while they are running \(Citrix, DB2, ODBC, Oracle, Oracle Forms Server, SAP, Uniface, Winsock, or WWW only\)](#)

You can instruct the Conductor to generate and save details about the script execution of selected virtual users by enabling Logfile Generation before you run your test.

To enable Logfile Generation:

1. On the Conductor's Script Assignment tab, highlight the script you want to monitor.
2. In the Debug Options column, click the browse (...) button (note that it might not be visible until you click it).
3. On the Debug Options dialog box, in the Logfile Generation Virtual User Range area, choose which virtual users (if any) to monitor. You can choose None or All Virtual Users, or choose Virtual User(s) and then type the numbers assigned to the virtual users you want to monitor. You can monitor individual virtual users or ranges of virtual users.
4. Click **OK** to save your changes.
5. From the Conductor's main menu, click **File>Save** to save your test session ID.
6. Run your test as usual. Each virtual user for which you enabled Logfile Generation will create a file containing information about their performance. After the test is finished, the Conductor will request all log files from the Players and save them in the directory `\Program Files\Compuware\QALoad\LogFiles` on the workstation where the Conductor is installed. Log files are named `<scriptname>_<middleware>_vu<AbsoluteVirtualUserNumber>.<ext>`, where:
 - ! `<scriptname>` is the name of the script the virtual user ran
 - ! `<middleware>` is the name of your middleware application
 - ! `<AbsoluteVirtualUserNumber>` is the identification number assigned to the virtual user
 - ! `<.ext>` is the file extension, dependent upon which middleware application you are testing. File extensions are listed in the following table:

Middleware	File Extension
Uniface WWW	.cap — A standard log file containing information about all statements executed during a test.
Citrix DB2 ODBC Oracle Oracle Forms Server SAP Winsock WWW	.log — A standard log file containing information about all statements executed during a test.

[View automatically generated log files from failed virtual users \(Oracle and WWW only\)](#)

A Player that fails to execute a test script automatically generates a log file (with .rip extension) that details the requests that were played back by that Player. You can view a Player's .rip file from the Conductor's Runtime Window while a test is running, or later in QALoad Analyze. At the end of a test, all .rip files are sent from the Players to the `\QALoad\LogFiles` directory and added to the merged timing file, where you can view them in Analyze.

To view a .rip file while the test is running:

1. In the Runtime Window's tree-view, right-click on a failed Player icon.
2. From the context menu, choose **Display RIP File**.

A new tab opens in the Runtime Window displaying details about the script the Player attempted to run, and which subsequently failed.

To view a .rip file in QALoad Analyze:

At the end of your test, after all test results have been collected, open your resulting timing file in Analyze. Click on the RIP Files group to see a list of all .rip files that were collected during the test.

Running a test

Running a load test

When you have a load test properly set up, you can start the test by clicking Run from the toolbar or by choosing Run>Start Test from the menu.

 **Tip:** While a test is running, the [Conductor's interface changes](#) to provide you with real-time test options.

Running a Series of Tests

You can also run a series of tests — a batch test. A batch test is comprised of multiple session ID files. When you run a batch test, the session files are executed sequentially until all of them are executed. The Conductor allows you to run multiple batch tests without operator intervention.

Checking out virtual user licenses

If you are licensed to run multiple copies of the Conductor, for example so different work groups have access to QALoad, you can check out virtual user licenses before running a load test to ensure that enough are available for your test run.

If you do not choose to check out your licenses before starting a test, QALoad will prompt you after you start the test and will attempt to check out the appropriate number of licenses. We recommend, though, that you check your licenses out manually before starting so you can be sure you have enough virtual users available before beginning your test run.

To check out virtual user licenses:


1. From the Conductor menu, select **Tools>Licensing**. The License Information dialog box opens.
 - If you are licensed for concurrent licensing (multiple Conductors) the Conductor will query your license server to determine how many licenses are currently available, and return the results to this dialog box. Go to step 2.
 - If you have a node-locked license (a single Conductor), then most of the options on this dialog box will be unavailable, as you will not need to, or be able to, check out virtual user licenses. All virtual users for which you are licensed will be available to only this Conductor. Click Close to return to your test setup.
2. In the Licensing Operations area, type how many virtual user licenses you want to check out in the **Number of Licenses** field.
3. Click **Check Out**. The licenses will be checked out to your Conductor, and unavailable to any other Conductor workstations on the network.


When you are done using your licensed virtual users, check them back in so they are once again available to other Conductor workstations on your network.

To check in virtual user licenses:

1. From the Conductor menu, choose **Tools>Licensing**. The License Information dialog box opens.
2. If you have licenses checked out, the Check in Virtual User License option is automatically selected for you.
3. Click **Check In**. The licenses will be made available to other Conductor workstations on the network.

Starting a test

Click Run  or choose Start Test from the Conductor's Run menu.

 **Note:** While a test is running, the toolbar changes to display the [Runtime Toolbar buttons](#).

Starting the Conductor from the command line


To start the Conductor from the command line, type:

```
mpwin32 <sessionID_file_name> /l /e /t
```

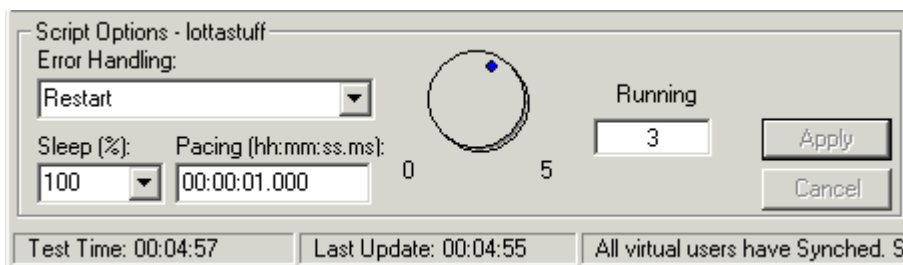
where valid startup parameters are:

Parameter	Description										
/l (Optional)	Creates a log file showing error messages and test status.										
/e (Optional)	Executes the indicated session ID file. If no session ID file is specified, it will simply launch the Conductor without opening a session.										
/t (Optional)	<p>Executes the Conductor at a set time or a set date and time. Time can be specified by either 12-hour or 24-hour format. The following examples of the /t parameter demonstrate each scenario.</p> <table border="1"> <thead> <tr> <th>Command</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>/t"03:30:00 pm"</td> <td>Starts the Conductor at 3:30PM today.</td> </tr> <tr> <td>/t06:00:00</td> <td>Starts the Conductor at 6:00AM today.</td> </tr> <tr> <td>/t"12/01/04 03:30:00 pm"</td> <td>Starts the Conductor on December 1, 2004 at 3:30PM.</td> </tr> <tr> <td>/t"12/25/04 14:00:00"</td> <td>Starts the Conductor on December 25, 2004 at 2:00PM.</td> </tr> </tbody> </table>	Command	Result	/t"03:30:00 pm"	Starts the Conductor at 3:30PM today.	/t06:00:00	Starts the Conductor at 6:00AM today.	/t"12/01/04 03:30:00 pm"	Starts the Conductor on December 1, 2004 at 3:30PM.	/t"12/25/04 14:00:00"	Starts the Conductor on December 25, 2004 at 2:00PM.
Command	Result										
/t"03:30:00 pm"	Starts the Conductor at 3:30PM today.										
/t06:00:00	Starts the Conductor at 6:00AM today.										
/t"12/01/04 03:30:00 pm"	Starts the Conductor on December 1, 2004 at 3:30PM.										
/t"12/25/04 14:00:00"	Starts the Conductor on December 25, 2004 at 2:00PM.										

Dialing up/down virtual users

 **Note:** If you haven't configured a ramp-up session, you will not be allowed to add or suspend virtual users while the test is running. For information about configuring a ramp-up session, see [Configuring a ramp-up session](#).

When your test is running, the bottom of the Test Information window turns into the dockable Runtime Control Panel, a portion of which is shown below:



If you click on a Player or script icon in the test's tree-view, the Runtime Control Panel will indicate how many virtual users are currently running on the selected Player machine or script. You can change the number of running virtual users per script or per Player by selecting the appropriate script or Player machine in the tree-view, and then typing a new number in the Running field (or by using the dial control).

To dial up or down (add or subtract) virtual users during a test:

1. When your test is running, click on the script or Player workstation in the Runtime Window's tree-view for which you want to add or subtract virtual users. The Running column in the top pane shows how many virtual users are currently running on that script or Player.
2. In the Runtime Control Panel, type a new number in the **Running** field or drag the dial control to change the number.
3. When you are done, click the **Apply** button. The Conductor will release or suspend the specified number of virtual users.

Your change do not take effect until you click the Apply button.

Increase/decrease runtime timing updates

While a test is running, you can change the frequency at which timing updates are sent from the Players to the Conductor. Decreasing the update interval will reduce the amount of overhead incurred in large load tests due to the communications between the Conductor and large numbers of virtual users.

On the Runtime Control Panel (bottom pane), choose from the following options:


No Updates: Choose this option to stop sending timing data while the test is running. Data will still be collected at the end of the test.

Send All: Choose this option to send all timing data as it is compiled.

Periodic Updates: Choose this option to specify a time interval for sending updates, then type the time interval (in seconds) below.

Any change takes effect immediately, and applies to all scripts in the test.

Manually Aborting a Test

To manually abort a test (stop script execution for all workstations), click on the Abort toolbar button .

Remove used datapool records after a test

You can remove used datapool records after a test by setting the Strip Datapool function before you run the test. Use this function when running a test where you have data in the external datapool which can only be used once by one virtual user at a time. (For example, when running transactions that have unique data constraints.) When activated, the Strip Datapool function will mark each piece of data in the datapool that is used during your test. When the test is over, the Strip Datapool function prompts you to remove the identified used data from the datapool. If you run the test again, only new data will be used for your subsequent test.

To use the Strip Datapool function:

1. With the current test's session ID file open in the Conductor, select the Script Assignment tab.
2. Click the **External Data** button. The External Data dialog box appears.
3. In the Central Datapool area, select the **Strip** check box. Click **OK**.
4. At the end of your test, a Strip Datapools prompt will appear asking if you wish to go to the Strip Datapools screen. Click **Yes**.
5. The Strip Data Pool dialog box appears. Click the **Strip** button.
6. When you are finished, click **Done**.



Running a series of tests (batch)

Running a batch test

1. Select **Run>Batch Test**.
2. Add the required session ID files to the Batch List using **Add** or **Load**.
3. Click **Start** to initiate the batch.

The Conductor then executes each of the session ID files in sequence.


Terminating a batch test


You can stop a batch of tests the same way you would stop a single session test, by clicking the Abort  or Exit  toolbar buttons.

Monitoring a running test


Watching a script execute

Use the Debug window in the Details view of the runtime Conductor to view the executing script. Note that it is possible that you won't see the execution of every statement, in order to minimize network traffic between the Conductor and the Players. The QALOAD.INI file's debug messages-per-sec parameter determines how frequently the Player sends its script debug status to the Conductor. At its default value of one message per second, the Player can execute several statements without sending a debug message to the Conductor.

 To open the Debug window, select a workstation in the global control window and click the Debug toolbar button.

 **Note:** The Conductor highlights the script line that it is currently executing.

Viewing datapool usage

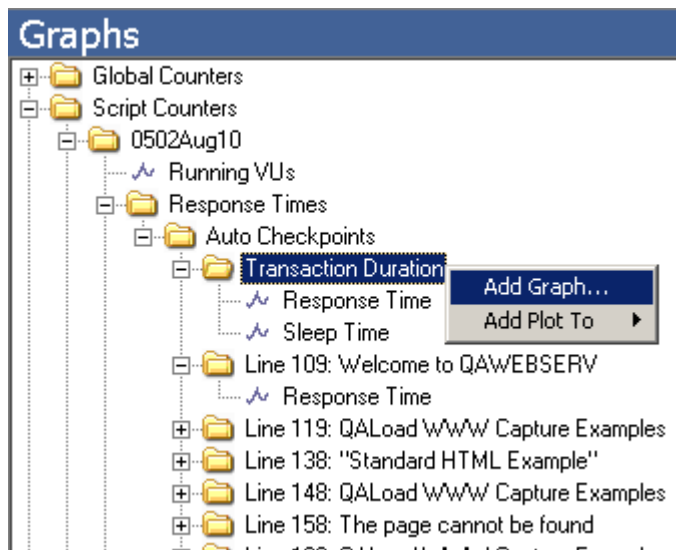
 Highlight a script in the global control window and click Datapool to open the Datapool window. As the script executes Get_Data commands, the Datapool window reflects the current datapool record being used by the script.

Graphing checkpoints

Use the Graphs view of the runtime Conductor to create real-time graphs of checkpoint response times during script execution. Similar graphs are also available for post-test analysis in QALoad Analyze.

Selecting checkpoints to graph

Before you can review checkpoint response times in graph form, you must select the checkpoint counters to include. Checkpoints are listed in the tree view on the left side of the Graphs view of the runtime Conductor, as shown in the example below. Both automatic and user-defined checkpoints appear in the Response Times folder of each running script.



Creating a graph of checkpoint response times

To choose a checkpoint that should appear in a graph, highlight the checkpoint name, right-click and choose either Add Graph to create a new graph or Add Plot To to add a data plot to an existing graph.

If you choose the Add Graph option, the **Add Graph dialog box** appears. Select the options for how the graph should appear and click OK.

To better identify problem checkpoints, you can set thresholds on plots or graphs that indicate the number of times the data record for that checkpoint has gone above or below the number you set. Thresholds can be set from the **Advanced tab of the Add Graph dialog box** or by right-clicking on an existing graph and choosing Thresholds.

Highlighting individual plots

If you create several plots on a single graph, it may become difficult to see individual plots. To increase a plot's visibility, click on a plot in the graph or a plot's number in the graph's legend. When highlighted, the plot appears thicker and darker on the graph.

Saving checkpoint graphs to a session ID

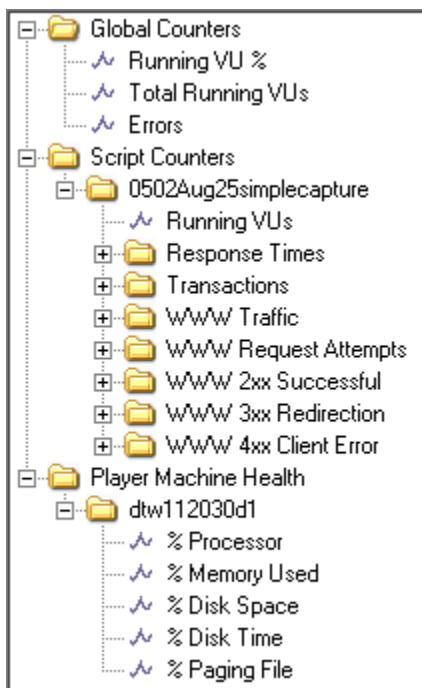
Checkpoint graphs that are created in the Conductor are automatically saved to the current session ID file. To remove all graphs you added, click Graph>Restore Default Graph Layout.

Graphing counter data

Use the Graphs view of the runtime Conductor to create real-time graphs of counter data during script execution. Similar graphs are also available for post-test analysis in QALoad Analyze.

Selecting counters to graph

All counter data that is available for graphing is located in the tree view on the left side of the of the Graphs view Data window, as shown below.



Scripts of any middleware type collect the following default counter data, which is available in the Conductor for real-time graphing:

- ! **Global counters:** Running VU%, total running VUs, and errors
- ! **Script counters:** Running VUs, response times, and transactions
- ! **Player machine health:** % processor, % memory used, % disk space, %disk time, % paging file

Additional middleware-based graphs are also generated by default and vary by middleware. For example, for the WWW middleware, several performance-based counters are automatically collected and available for graphing, including server responses and WWW traffic. You can monitor this data to determine the optimum rate of performance of the application that is running.

Graphing counter statistics

To choose a counter that should appear in a graph, highlight the checkpoint counter name or group of counters (folder), right-click and choose either Add Graph to create a new graph or Add Plot To to add a data plot to an existing graph.

If you choose the Add Graph option, the [Add Graph dialog box](#) appears. Select the options for how the graph should appear and click OK.

To better identify problems in the test, you can set thresholds on plots or graphs that indicate the number of times the data record for that counter has gone above or below the number you set. Thresholds can be set from the [Advanced tab of the Add Graph dialog box](#) or by right-clicking on an existing graph and choosing Thresholds.

Highlighting individual plots

If you create several plots on a single graph, it may become difficult to see individual plots. To increase a plot's visibility, click on a plot in the graph or a plot's number in the graph's legend. When highlighted, the plot appears thicker and darker on the graph.

Saving counter data graphs to a session ID


Counter data graphs that are created in the Conductor are automatically saved to the current session ID file. To remove all graphs you added, click Graph>Restore Default Graph Layout.

Recording and replaying a test

Recording and replay overview

As a load test is running, the Conductor records each event in a .REC file. After the load test completes, you can open the .REC file to replay the load test to evaluate important events that occurred, such as a sudden increase in processor usage.

When you replay the load test, the playback looks identical to the actual load test, with the exception of the playback toolbar, which appears at the top of the screen. When you open the .REC replay file in the Conductor, you can replay, fast-forward, double fast-forward, or pause.

 **Note:** The replay feature provides a visual re-enactment of the load test; it does not perform the actual test or connect to any servers.

Recording a test

Replaying a test

Recording a test

Load tests can be recorded by selecting the Record Load Test option from the Conductor. When this option is activated and the load test begins, a prompt appears for you to specify a file name for the recording. When the load test is completed, you can [replay the test](#).

To set the Conductor to record a load test:







1. On the **Test Information** tab of the [Conductor's main window](#), select the **Enable Test Recording** check box.
2. Start the load test. The **Record** dialog box appears.
3. Type a name for the record file. Click **OK**.


Replaying a test

If you set the Conductor to [record load tests](#), you can play them back after the test completes. Replaying a recorded load test does not perform the actual load test. A replay provides a visual re-enactment of the events that took place during the load test.

To replay a recorded load test:

1. From the Conductor's **Run** menu, choose **Test Recording>Replay a Load Test**. The **Open Record File** dialog box appears.
2. Browse for the recording file (*filename.rec*) that you saved when the load test started. Click **Open**.
3. The test will play back in a viewer that contains a playback toolbar. Use the toolbar buttons described in the following table to control the playback.

Button	Action
	Restarts the test replay from the beginning
	Replays the test at normal speed
	Replays the test twice as fast as normal
	Replays the test four times as fast as normal
	Pauses the replay at the current snapshot
	Exits test replay and opens the Conductor test setup window

 **Note:** Test control features such as dial-up/dial-down do not work during test replay, but the effects from these features can be observed in the replay. Also, virtual user error details on the Virtual User Info window are not available during replay. Detailed error information is available in the timing file and can be [viewed with Analyze](#).


Analyzing load test data

Analyzing load test data

By default, load test timing data is sent from the Conductor to Analyze at the end of a load test. Any appropriate server monitoring data is also sent to Analyze and merged into your timing file (.tim).

You can set an option in the Conductor to automatically launch Analyze at the end of a load test ([details](#)), or you can open Analyze manually from the Conductor toolbar or your QALoad program group.

Creating a timing file (.tim)

Once all workstations stop executing, click the Quit toolbar button  to complete the test and automatically create the timing file (.tim).

Viewing test statistics



Compute test statistics by choosing Launch Analyze from the Conductor's Tools menu or by clicking on the Analyze toolbar button.

Integration and server monitoring

Remote Monitoring

Remote Monitoring

Remote Monitoring allows you to extract data from Windows registry counters or SNMP counters (which can be Windows or UNIX) from the server(s) under stress without requiring any software to be installed on the server(s). However, to collect Windows registry counters, you must have a valid sign-on for the server(s) under test.

To use Remote Monitoring:

- ! You must have login access to the machines you want to monitor
- ! You must select the counters to monitor on the machines to monitor using the Conductor's [Machine Configuration tab](#)
- ! To collect SNMP counters, SNMP must be enabled on the Remote Monitor machine. Refer to your operating system help for information about enabling SNMP.

While your test is running, QALoad will collect the appropriate counter data and write it to your timing file where you can view it in Analyze after the test. [What counters are available?](#)

To set up Remote Monitoring, see [Setting up a test session](#).

Adding a Remote Monitor machine

You can add Remote Monitor machines to a test as part of your overall test session setup, or individually at any time. You do not need to install any additional components on the machines to monitor.

To set up a whole test session, see [Setting up a test session](#).

To add a Remote Monitor machine to a test session:

1. Open the appropriate test session ID.
2. On the Machine Configuration tab, select the Remote Monitor Machines option.
3. Click the **Enable Data Capture** check box.
4. Click the **New** button to configure a machine.
5. On the Remote Machine Configuration dialog box, type the host name of the machine to use as a Remote Monitor machine. Use a fully-qualified name in one of the following formats:

```
<machinename>\username
```

or

```
<domain>\username
```

6. In the Remote Machine Type area, choose whether to gather Windows Registry Counters or SNMP Counters (All platforms).
7. Click the **Test** button to check the connection to the remote machine. The Conductor will query the machine and display a message indicating if the connection is successful.
8. When you are finished, click **OK**.
9. On the Machine Configuration tab, choose the counters to monitor by clicking the check box next to the counter name.

Remote Monitoring counters

QALoad's Remote Monitor Agents can monitor Windows registry counters or SNMP counters.

Windows Registry Counters

Remote Monitoring Agents can monitor the same Windows registry counters as PERFMON, the performance monitoring application available with the Windows operating system. The Windows registry option monitors machines that run Windows 2000 and XP. To retrieve Windows Registry Counters, you must have access, via a user name and password, to the remote machine.

SNMP Counters

SNMP Remote Monitoring uses the SNMP service to provide network and system counters. SNMP counters can be retrieved from any machine that is running an SNMP server. Although SNMP does not require a user name and password, the SNMP agent must be configured to allow read-only access from the Conductor machine. SNMP counters that are supported by QALoad Remote Monitoring are categorized below.

ICMP

icmpInMsgs/sec: the rate at which ICMP messages are received
icmpInErrors: the number of ICMP messages received having ICMP errors
icmpInDestUnreachs: the number of ICMP Destination Unreachable messages received
icmpInTimeExcds: the number of ICMP Time Exceeded messages received
icmpInParmProbs: the number of ICMP Parameter Problem messages received
icmpInSrcQuenchs: the number of ICMP Source Quench messages received
icmpInRedirects/sec: the rate at which ICMP Redirect messages are received
icmpInEchos/sec: the rate at which ICMP Echo messages are received
icmpInEchoReps/sec: the rate at which ICMP Echo Reply messages are received
icmpInTimestamps/sec: the rate at which ICMP Timestamp messages are received
icmpInTimestampReps/sec: the rate at which ICMP Timestamp Reply messages are received
icmpInAddrMasks: the number of ICMP Address Mask Request messages received
icmpInAddrMaskReps: the number of ICMP Address Mask Reply messages received
icmpOutMsgs/sec: the rate at which ICMP messages are sent
icmpOutMsgs/sec: the number of ICMP messages not sent due to ICMP errors
icmpOutDestUnreachs: the number of ICMP Destination Unreachable messages sent
icmpOutTimeExcds: the number of ICMP Time Exceeded messages sent
icmpOutParmProbs: the number of ICMP Parameter Problem messages sent
icmpOutSrcQuenchs: the number of ICMP Source Quench messages sent
icmpOutRedirects/sec: the number of ICMP Redirect messages sent
icmpOutEchos/sec: the number of ICMP Echo messages sent
icmpOutEchoReps/sec: the number of ICMP Echo Reply messages sent
icmpOutTimestamps/sec: the number of ICMP Timestamp messages sent
icmpOutTimestampReps/sec: the number of ICMP Timestamp Reply messages sent
icmpOutAddrMasks: the number of ICMP Address Mask Request messages sent
icmpOutAddrMaskReps: the number of ICMP Address Mask Reply messages sent

IP

ipForwarding: the indication of whether this entity is acting as an IP router in respect to the forwarding of datagrams received by, but not addressed to, this entity.

ipDefaultTTL: the default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

ipInReceives/sec: the rate of input datagrams received from interfaces, including those received in error.

ipInHdrErrors: the number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, and so on.

ipInAddrErrors: the number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity.

ipForwDatagrams/sec: the rate of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination.

ipInUnknownProtos: the number of locally-addressed datagrams receive successfully but discarded because of an unknown or unsupported protocol.

ipInDiscards: the number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (for example, for lack of buffer space).

ipInDelivers/sec: the rate of input datagrams successfully delivered to IP user-protocols (including ICMP).

ipOutRequests: the number of IP datagrams which local IP user-protocols (including ICMP) supplied to IP in requests for transmission.

ipOutDiscards: the number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (for example, for lack of buffer space).

ipOutNoRoutes: the number of IP datagrams discarded because no route could be found to transmit them to their destination.

ipReasmTimeout: the maximum number of seconds which received fragments are held while they are awaiting reassembling at this entity.

ipReasmReqds: the number of IP fragments received which needed to be reassembled at this entity.

ipReasmOKs: the number of IP datagrams successfully re-assembled.

ipReasmFails: the number of failures detected by the IP re-assembly algorithm (for whatever reason: timed out, errors, etc).

ipFragOKs: the number of IP datagrams that have been successfully fragmented at this entity.

ipFragFails: the number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, for example, because their Don't Fragment flag was set.

ipFragCreates/sec: the rate of IP datagram fragments that have been generated as a result of fragmentation at this entity.

ipRoutingDiscards: the number of routing entries which were chosen to be discarded even though they are valid.

SNMP

snmpInPkts/sec: the rate of messages delivered to the SNMP entity from the transport service.

snmpOutPkts/sec: the rate at which SNMP Messages were passed from the SNMP protocol entity to the transport service.

snmpInBadVersions: the number of SNMP messages which were delivered to the SNMP entity and were for an unsupported SNMP version.

snmpInBadCommunityNames: the number of SNMP messages delivered to the SNMP entity which used a SNMP community name not known to said entity.

snmpInBadCommunityUses: the number of SNMP messages delivered to the SNMP entity which represented an SNMP operation which was not allowed by the SNMP community named in the message.

snmpInASNParseErrs: the number of ASN.1 or BER errors encountered by the SNMP entity when decoding received SNMP messages.

snmpInTooBig: the number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is tooBig.

snmpInNoSuchNames: the number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is noSuchName.

snmpInBadValues: the number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is badValue.

snmpInReadOnlys: the number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is readOnly.

snmpInGenErrs: the number of SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the error-status field is genErr.

snmpInTotalReqVars/sec: the rate of MIB objects which have been retrieved successfully by the SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs.

snmpInTotalSetVars/sec: the rate of MIB objects which have been altered successfully by the SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs.

snmpInGetRequests/sec: the rate of SNMP Get-Request PDUs which have been accepted and processed by the SNMP protocol entity.

snmpInGetNexts/sec: the rate of SNMP Get-Next PDUs which have been accepted and processed by the SNMP protocol entity.

snmpInSetRequests/sec: the rate of SNMP Get-Response PDUs which have been accepted and processed by the SNMP protocol entity.

snmpInGetResponses/sec: the rate of SNMP Set-Request PDUs which have been accepted and processed by the SNMP protocol entity.

snmpInTraps: the number of SNMP Trap PDUs which have been accepted and processed by the SNMP protocol entity.

snmpOutTooBig: the number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is tooBig.

snmpOutNoSuchNames: the number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status is noSuchName.

snmpOutBadValues: the number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is badValue.

snmpOutGenErrs: the number of SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the error-status field is genErr.

snmpOutGetRequests/sec: the rate of SNMP Get-Request PDUs which have been generated by the SNMP protocol entity.

snmpOutGetNexts/sec: the rate of SNMP Get-Next PDUs which have been generated by the SNMP protocol entity.

snmpOutSetRequests/sec: the rate of SNMP Set-Request PDUs which have been generated by the SNMP protocol entity.

snmpOutGetResponses/sec: the rate of SNMP Get-Response PDUs which have been generated by the SNMP protocol entity.

snmpOutTraps: the number of SNMP Trap PDUs which have been generated by the SNMP protocol entity.

snmpOutTraps: indicates whether the SNMP entity is permitted to generate authenticationFailure traps.

TCP

tcpRtoAlgorithm: the algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

tcpRtoMin: the minimum value permitted by a TCP implementation for the retransmission timeout.

tcpRtoMax: the maximum value permitted by a TCP implementation for the retransmission timeout.

tcpMaxConn: the limit on the total number of TCP connections the entity can support.

tcpActiveOpens: the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.

tcpAttemptFails: the number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.

tcpEstabResets: the number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.

tcpCurrEstab: the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.

tcpInSegs/sec: the rate at which segments are received, including those received in error.

tcpOutSegs/sec: the rate at which segments are sent, including those on current connections but excluding those containing only retransmitted octets.

tcpRetransSegs/sec: the rate at which segments are retransmitted.

tcpInErrs/sec: the rate at which segments are received in error.
 tcpOutRsts/sec: the rate at which segments containing the RST flag are sent.
 tcpPassiveOpens: the total number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.

UDP

udpInDatagrams/sec: the rate of UDP datagrams being delivered to UDP users.
 udpNoPorts/sec: the rate of received UDP datagrams for which there was no application at the destination port.
 udpInErrors: the number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
 udpOutDatagrams/sec: the rate at which UDP datagrams are sent.

Solaris: Sun System

Collisions/sec: the rate of output collisions.
 CpuUser%: the percentage of non-idle processor time that is spent in user mode.
 CpuNice%: the percentage of non-idle processor time that is spent in nice mode.
 CpuSys%: the percentage of non-idle processor time that is spent in system mode.
 CpuIdle%: the percentage of idle processor time.
 IfInPackets/sec: the rate of input packets.
 IfOutPackets/sec: the rate of output packets.
 IfInErrors: the total number of input errors.
 IfOutErrors: the total number of output errors.
 Interrupts/sec: the rate of system interrupts.
 PagesIn KBytes/sec: the rate of pages read in from disk.
 PagesOut KBytes/sec: the rate of pages written to disk.
 SwapIn KBytes/sec: the rate at which pages are being swapped in.
 SwapOut KBytes/sec: the rate at which pages are being swapped out.

HP-UX: HP System

AvgJobs1: the average number of jobs in the last minute * 100.
 AvgJobs5: the average number of jobs in the last 5 minutes * 100.
 AvgJobs15: the average number of jobs in the last 15 minutes * 100.
 CpuUser%: the percentage of non-idle processor time that is spent in user mode.
 CpuNice%: the percentage of non-idle processor time that is spent in nice mode.
 CpuSys%: the percentage of non-idle processor time that is spent in system mode.
 CpuIdle%: the percentage of idle processor time.
 FreeMemory KBytes: the amount of idle memory.
 FreeSwap KBytes: the amount of free swap space on the system.
 MaxProc: the maximum number of processes allowed.
 MaxUserMem KBytes: the amount of maximum user memory on the system.
 PhysMemory KBytes: the amount of physical memory on the system.
 Users: the number of users logged on to the machine.

LINUX Memory

AvailableSwap KBytes: the available swap on the system.
 Buffered KBytes: the amount of memory used as buffers.
 Cached KBytes: the amount of memory cached.
 FreeMemory KBytes: the amount of idle memory.
 Shared KBytes: the amount of memory shared.
 TotalMemory KBytes: the total amount of memory on the system.
 TotalSwap KBytes: the total swap size for the system.

LINUX System

CpuUser%: the percentage of non-idle processor time that is spent in user mode.
CpuNice%: the percentage of non-idle processor time that is spent in nice mode.
CpuSys%: the percentage of non-idle processor time that is spent in system mode.
CpuIdle%: the percentage of idle processor time.

Windows HTTP Server

httpTotalFilesSent: the total number of files sent by this HTTP server.
httpTotalFilesReceived: the total number of files received by this HTTP server.
httpCurrentAnonymousUsers: the number of anonymous users currently connected to this HTTP server.
httpCurrentNonAnonymousUsers: the number of non-anonymous users currently connected to this HTTP server.
httpTotalAnonymousUsers: the total number of anonymous users that have ever connected to this HTTP server.
httpTotalNonAnonymousUsers: the total number of non-anonymous users that have ever connected to this HTTP server.
httpMaximumAnonymousUsers: the maximum number of anonymous users simultaneously connected to this HTTP server.
httpMaximumNonAnonymousUsers: the maximum number of non-anonymous users simultaneously connected to this HTTP server.
httpCurrentConnections: the current number of connections to the HTTP server.
httpMaximumConnections: the maximum number of simultaneous connections to the HTTP server.
httpConnectionAttempts: the total number of connection attempts to the HTTP server.
httpLogonAttempts: the total number of logon attempts to the HTTP server.
httpTotalOptions: the total number of requests made to this HTTP server using the OPTIONS method.
httpTotalGets: the total number of requests made to this HTTP server using the GET method.
httpTotalPosts: the total number of requests made to this HTTP server using the POST method.
httpTotalHeads: the total number of requests made to this HTTP server using the HEAD method.
httpTotalPuts: the total number of requests made to this HTTP server using the PUT method.
httpTotalDeletes: the total number of requests made to this HTTP server using the DELETE method.
httpTotalTraces: the total number of requests made to this HTTP server using the TRACE method.
httpTotalMove: the total number of requests made to this HTTP server using the MOVE method.
httpTotalCopy: the total number of requests made to this HTTP server using the COPY method.
httpTotalMkcol: the total number of requests made to this HTTP server using the MKCOL method.
httpTotalPropfind: the total number of requests made to this HTTP server using the PROPFIND method.
httpTotalProppatch: the total number of requests made to this HTTP server using the PROPPATCH method.
httpTotalSearch: the total number of requests made to this HTTP server using the MS-SEARCH method.
httpTotalLock: the total number of requests made to this HTTP server using the LOCK method.
httpTotalUnlock: the total number of requests made to this HTTP server using the UNLOCK method.
httpTotalOthers: the total number of requests made to this HTTP server not using the OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, MOVE, MKCOL, PROPFIND, PROPPATCH, MS-SEARCH, LOCK or UNLOCK methods.
httpCurrentCGIRequests: the number of Common Gateway Interface requests currently being serviced by this HTTP server.
httpCurrentBGIRequests: the number of Binary Gateway Interface requests currently being serviced by this HTTP server.
httpTotalCGIRequests: the total number of Common Gateway Interface requests made to this HTTP server.
httpTotalBGIRequests: the total number Binary Gateway Interface requests made to this HTTP server.
httpMaximumCGIRequests: the maximum number of Common Gateway Interface requests simultaneously processed by this HTTP server.
httpMaximumBGIRequests: the maximum number of Binary Gateway Interface requests simultaneously processed by this HTTP server.

httpCurrentBlockedRequests: the current number of requests being temporarily blocked by this HTTP server.
 httpTotalBlockedRequests: the total number of requests that have been temporarily blocked by this HTTP server.
 httpTotalRejectedRequests: the total number of requests that have been rejected by this HTTP server.

Windows FTP Server

ftpTotalFilesSent: the total number of files sent by this FTP server.
 ftpTotalFilesReceived: the total number of files received by this FTP server.
 ftpCurrentAnonymousUsers: the number of anonymous users currently connected to this FTP server.
 ftpCurrentNonAnonymousUsers: the number of non-anonymous users currently connected to this FTP server.
 ftpTotalAnonymousUsers: the total number of anonymous users that have ever connected to this FTP server.
 ftpTotalNonAnonymousUsers: the total number of non-anonymous users that have ever connected to this FTP server.
 ftpMaximumAnonymousUsers: the maximum number of anonymous users simultaneously connected to this FTP server.
 ftpMaximumNonAnonymousUsers: the maximum number of non-anonymous users simultaneously connected to this FTP server.
 ftpCurrentConnections: the current number of connections to the FTP server.
 ftpMaximumConnections: the maximum number of simultaneous connections to the FTP server.
 ftpConnectionAttempts: the total number of connection attempts to the FTP server.
 ftpLogonAttempts: the total number of logon attempts to the FTP server.

Server Analysis agents

Server Analysis agents

Server Analysis agents use enhanced ServerVantage technology to provide server utilization data without a complete ServerVantage deployment. Server Analysis agents, provided on the QACenter Performance Edition CD, are quickly and easily installed on the servers that you wish to monitor during a load test. Server Analysis agents provide you with valuable server utilization metrics — called counters — on Web servers, application servers, and database servers being exercised by your load test to help you to pinpoint performance bottlenecks when load testing.

Unlike a full ServerVantage installation, you can start, stop, and configure Server Analysis Agents right from the familiar interface of the QALoad Conductor.

Server utilization data from the agents and response time information from QALoad is all automatically downloaded and correlated through the use of ActiveAnalysis, and is available for post-test analysis through QALoad Analyze.

For details about how to use Server Analysis agents in a load test, see [Setting up Server Analysis agents](#).

Setting up Server Analysis agents

This procedure assumes you are licensed to use the Server Analysis agents.

To set up a Server Analysis agent:

1. Open a test session in QALoad Conductor.
2. On the **Machine Configuration** tab, click the **Server Analysis agents** option.

3. Click the **Discover Machines** button to locate all workstations on your network with QALoad Players and Server Analysis agents installed. All available servers are listed. (Compuware recommends that you choose to install the Server Analysis agent only on those machines you need to monitor during a load test.)
4. Select the **Enable Data Capture** check box to enable Server Analysis agent monitoring.
5. Discover which counters are available on your workstations:
 - Individually: Double-click on an individual Server Analysis agent. The counters available on that server will be listed.
 - Globally: Click the Discover All Agent Counters button. All available Server Analysis agents will be queried and the counters available on each server will be listed. A progress bar will note how many agents are available, how many have been queried, and how many (if any) failed to return counter information.
6. After discovering all available counters, choose which Server Analysis agents to monitor during your test by selecting the check box next to the server name.
7. Choose which counters to include:
 - By default, when you select a Server Analysis agent to monitor, all its available counters are also selected for monitoring. You can individually de-select any counters you don't wish to include.
 - Alternately, you can select QALoad-provided templates of counters based upon your particular server's setup. To do so:
 - Click the Add Templates to Selected Agent button.
 - On the QALoad – Select Templates dialog box, select the template to apply (for example, Server Health) and click OK.

Application Expert/ApplicationVantage


Overview

Application Expert is a Windows-based tool that enables you to examine the effects the network will have on the performance of new or modified applications prior to live deployment. Application Expert provides granular thread details that allow network managers to identify poorly performing applications. QALoad integrates with Application Expert versions 8.0 and 9.0 and ApplicationVantage 9.3 to help you analyze network performance during a load test. QALoad also provides test data that you can open in both Application Expert and ApplicationVantage.

Before QALoad can collect network data during a load test, the following must be true:

- ! The ApplicationVantage Agent is installed on the same machine as the QALoad Conductor. You can install either the ApplicationVantage Agent or the ApplicationVantage Remote Agent.
- ! The QALoad Conductor and Player machines are located on the same LAN.
- ! You have set up IP address pairs in the Conductor's Application Expert section of the Machine Configuration tab.
[How?](#)

At test time, the Conductor will automatically start the Application Vantage Agent. At the end of the test, the Conductor will stop the agent and return captured information to a trace file you can open in Application Expert or in ApplicationVantage. If either tool is installed on the Conductor machine, QALoad will generate an XML report that details the network traffic that was captured.

-  **Hint:** For information about Application Expert or ApplicationVantage, refer to the documentation you received with your purchase of these tools.

Overview


Application Expert is a Windows-based tool that enables you to examine the effects the network will have on the performance of new or modified applications prior to live deployment. Application Expert provides

granular thread details that allow network managers to identify poorly performing applications. QALoad integrates with Application Expert versions 8.0 and 9.0 and ApplicationVantage 9.3 to help you analyze network performance during a load test. QALoad also provides test data that you can open in both Application Expert and ApplicationVantage.

Before QALoad can collect network data during a load test, the following must be true:

- ! The ApplicationVantage Agent is installed on the same machine as the QALoad Conductor. You can install either the ApplicationVantage Agent or the ApplicationVantage Remote Agent.
- ! The QALoad Conductor and Player machines are located on the same LAN.
- ! You have set up IP address pairs in the Conductor's Application Expert section of the Machine Configuration tab.
[How?](#)

At test time, the Conductor will automatically start the Application Vantage Agent. At the end of the test, the Conductor will stop the agent and return captured information to a trace file you can open in Application Expert or in ApplicationVantage. If either tool is installed on the Conductor machine, QALoad will generate an XML report that details the network traffic that was captured.


-  **Hint:** For information about Application Expert or ApplicationVantage, refer to the documentation you received with your purchase of these tools.

Setting up IP address pairs from the Conductor

To use the ApplicationVantage Agent to collect data for Application Expert or ApplicationVantage, it is necessary to set up IP address pairs to monitor from the Conductor.

To set up IP address pairs:

1. With the session ID you want to use for your test open, click on the Conductor's **Machine Configuration** tab.
2. Click the **Discover Machines** button for the Conductor to query your test network for installed Player Agents. If the **Discover Machines** button isn't available, select the **Player Agents** option button and then try again. The **Machine Configuration** tab will be populated with names of available Player machines.
3. Select the **Application Expert** option button to access the Application Expert-specific fields.
4. In the **Machines to Add** area, choose the Player machine that will be running the virtual user to be captured. To make your selection, simply highlight the machine name and click the arrow button to move the machine name into the **IP Pairs** area as Address 1. Address 2 should be the IP or machine name of the first tier of the application being tested, or you can simply type *any* to capture all traffic for Address 1. Select the **Include** check box to monitor communications between those two machines during your load test.

 **Note:** If the network environment is running over a switch, then Address 1 must be the IP of the Conductor machine and the virtual user to capture must execute from that machine. If the network environment is running over a hub (shared network), then Address 1 can be any Player machine available for testing. For more information, see [Configuring a test to use Application Expert](#).

5. The **Login Area** section should only be modified if you have installed the ApplicationVantage Remote Agent and not the ApplicationVantage Agent. These fields will contain the network information for the workstation where your ApplicationVantage Remote Agent is installed.
 - a. **Host Name:** This field automatically lists the name of the machine where your Conductor is installed. Do not change this.
 - b. **Username:** Type the user name used during the installation of the ApplicationVantage Remote Agent. If using the ApplicationVantage Agent, type the username Admin.
 - c. **Password:** Type the password used during the installation of the ApplicationVantage Remote Agent. If using the ApplicationVantage Agent, type the password Admin.
 - d. **NIC Name:** From the drop-down list, select the NIC (network interface card) that is used by the machine in the Host Name field.
6. Save this information to your test session ID file by clicking **File>Save**.

7. (Optional) To save the current machine setup for re-use, create a new configuration file (.cfg). [How?](#)
8. Run your test. At test time, the Conductor passes this information to your installation of Application Expert, which then captures the communications between the specified pairs of machines and saves that information to a file you can open in Application Expert after the test has finished. The file, named `sessionname_date_time.opt` (Application Expert version 8.0) or `sessionname_date_time.opx` (Application Expert version 9.0), is saved to `Program Files\Compuware\QALoad\LogFiles`.

Configuring a test to use Application Expert

There are two possible configurations when using the Application Expert component of QALoad. The type of network where QALoad is installed will determine which configuration you will use. In either instance, the test will have a stand-alone virtual user, which we will call an Expert User as this is the virtual user to be captured by Application Expert. The Expert User will run from a Player machine by itself – no other virtual users will execute on this machine. The remaining virtual users will be executing from other Player machines within the test environment. The two configurations are described below.

Hub or shared network

In a shared network, Application Expert has the ability to see network traffic from any device connected to a hub. When using QALoad in this environment, the test can be configured to have multiple Player machines – with one of those machines running the Expert User. To capture the network activity of the Expert User, set up Address 1 of the IP pairs in the Conductor's Machine Configuration tab to point to this machine.

Switched network

In a switched network, Application Expert will only see the network traffic of the device where it is installed – which will be the QALoad Conductor machine in this case. When using QALoad in this environment, the test must be configured to run the Expert User from a Player installed on the same machine as the Conductor. All remaining virtual users must run from Player machines other than the Conductor machine. To capture the network activity of the Expert User, set up Address 1 of the IP pairs in the Conductor's Machine Configuration tab to point to the Conductor machine. Please note that the Player Agent needs to be installed and running on the Conductor machine.

ServerVantage

Server monitoring with ServerVantage

If you are currently a licensed user of Compuware's ServerVantage, you can integrate data from your existing ServerVantage deployment directly into a QALoad timing file at the end of a load test.

For this method to be successful, the following conditions must be met:

- ! ServerVantage must be installed and configured correctly on your system
- ! ServerVantage must be scheduled to monitor the specified performance counters at a time that coincides with a running QALoad test
- ! QALoad must be able to access the appropriate Agent stations to collect resource utilization data at the end of the load test.

ServerVantage

ServerVantage (formerly EcoTOOLS) monitors the availability and performance of applications, databases and servers, allowing users to centrally manage events across all application components— Web servers, firewalls, application servers, file systems, databases, middleware, and operating systems. ServerVantage simultaneously monitors these components, analyzes both historical and real-time events, and correlates monitored information for problem detection.

Integration with ServerVantage is configured from the QALoad Conductor. Performance counters collected during a load test are included in the test's timing file and can be sorted and displayed in QALoad Analyze in much the same way as QALoad timing data. For more information about installing or configuring ServerVantage refer to its product documentation.

Setting up integration with ServerVantage

For instructions for integrating with ServerVantage, read the appropriate section below:

To set up integration with ServerVantage:

1. On the Conductor Test Information Screen, click the Monitoring Options tab.
2. Click the **Enable ServerVantage Integration** check box.
3. In the **Control Server Database Host** field, type the hostname of the machine where the ServerVantage server is located.
4. In the **Username** field, type a valid user name to access the ServerVantage server.
5. In the **Password** field, type the password that corresponds to the user name above.
6. Select the **Override Default Database** check box to provide the ServerVantage database name. When this option is not selected, QALoad uses the default ServerVantage database name. If you provided a different name during the installation of ServerVantage, select this option and type the name in the **Database Name** field.
7. In the Vantage Agent Configuration area, type the hostname of a machine(s) where a ServerVantage Agent is installed, and click the **Add** button to add it to your load test.

Troubleshooting

Conductor pre-test checks

Before a test begins, the Conductor completes the following pre-test checks of the parameter files and Players. If any of these checks fail, the Conductor displays an error message.

- ! Are there enough Players configured to support the number of users specified in the session ID file?
- ! Does the number of users specified in the session ID file exceed the maximum number of users defined by your authorization key?
- ! Can the specified compiled script files be accessed?
- ! Are all Players communicating with the Conductor? (The Conductor sends a request message to all the Players to verify that they are up and running.)

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 `\Program Files\Compuware\QALoad\Certificates` directory to the same default directory on the Player machine.

Tips for running QALoad tests on UNIX systems

To successfully run large QALoad tests on UNIX systems, you may need to make adjustments to your settings as described below:

General (AIX, Solaris, HP-UX, and RedHat Linux)

When you attempt to run a large number of virtual users on UNIX platforms, the virtual users do not always synch. If virtual users do not synch, try increasing the Virtual User Startup Delay. By default, QALoad Conductor sets the VU Startup Delay to 1 millisecond. This default is not high enough for UNIX platforms. If the UNIX Player receives a value less than 15 milliseconds, the delay will be 15 milliseconds or more.

To increase the delay:

1. In the QALoad Conductor, click **Tools>Options**.
2. Click the **Player** tab.
3. In the **VU Startup Delay** field, type the number of milliseconds to delay virtual user startup.

Solaris

The default file descriptor limit on Solaris has a "soft" limit of 64, and a "hard" limit of 1024 (Solaris 2.6). Per the Solaris 2 FAQ (refer to <http://www.wins.uva.nl/pub/solaris/solaris2.html>), the file descriptor limit is described in the getrlimit() manual page as: "One more than the maximum value that the system may assign to a newly created descriptor. This limit constrains the number of file descriptors that a process may create."

To increase this limit, system administrators can modify the `/etc/system` file and reboot the system. For example:

```
* set hard limit on file descriptors
set rlim_fd_max = 4096
*set soft limit on file descriptors
set rlim_fd_cur = 1024
```

HP-UX

On HP-UX, the default thread-per-process limit is 64. This means that only 58 virtual users per Player will sync. To increase the thread-per-process limit, the system administrator can use the System Administration Manager (SAM) to re-configure and to compile the kernel by selecting Kernel Configuration>Configurable Parameters>max_thread_proc. Increase the value to at least 2048.

On HP-UX 11i, the total number of threads default is 5000. Increase this by changing the nkthread Kernel parameter.

Timing file is too big

Depending on the length of the load test and the amount of data that was collected, timing files can grow to excessively large sizes that become difficult to handle. To prevent timing files from becoming too large, try modifying the following settings:

- ! Disable automatic middleware checkpoint timings in the Conductor
- ! Use the Conductor's timing data thinning options

Both of these settings are located on the **Timing Options dialog box**, which can be accessed from the **Script Assignment tab** of the Conductor.

Oracle Forms Server playback error codes

QALoad displays error codes during playback for specific exception messages. While debugging, refer to the table below that lists error codes and descriptions that apply to Oracle Forms Server scripts.

Most of the errors listed below are client request errors due to JVM memory issues. When the error is due to a server problem, the error message indicates a connection issue or a bad response from the server. All these errors cause playback to fail. When the error is client-related, you can work around the JVM memory issue by tweaking the Player machine's [Threads Per Player value](#) in QALoad Conductor. When the error is server-related, the server is unable to handle the load. The server typically throws out connection requests, does not respond to requests, or terminates connections during playback.

Error code	Description
OFS-ERROR-001	Failed to create the replay log file.
OFS-ERROR-002	Failed while processing server detail message. Unknown control handle.
OFS-ERROR-003	Failed to send a heartbeat message.
OFS-ERROR-004	OracleAppsLogin: Error: icx_ticket not found in OracleAppsLogin, please check URL, userid, and password.
OFS-ERROR-005	Failed to set Boolean property of a RunForm object.
OFS-ERROR-006	Failed to process Boolean property of a RunForm object.
OFS-ERROR-007	Failed to set Point property of a RunForm object.
OFS-ERROR-008	Failed to process point property of a RunForm object.
OFS-ERROR-009	Failed to set Byte property of a RunForm object.
OFS-ERROR-010	Failed to process Byte property of a RunForm object.
OFS-ERROR-011	Failed to set Integer property of a RunForm object.
OFS-ERROR-012	Failed to process Integer property of a RunForm object.
OFS-ERROR-013	Failed to set String property of a RunForm object.
OFS-ERROR-014	Failed to process String property of a RunForm object.
OFS-ERROR-015	Failed to set Void property of a RunForm object.
OFS-ERROR-016	Failed to process Void property of a RunForm object.
OFS-ERROR-017	Failed to process Character property of a RunForm object.
OFS-ERROR-019	Failed to process Float property of a RunForm object.
OFS-ERROR-020	Failed to set Date property of a RunForm object.
OFS-ERROR-021	Failed to process Date property of a RunForm object.

OFS-ERROR-022	Failed to set Rectangle property of a RunForm object.
OFS-ERROR-023	Failed to process Rectangle property of a RunForm object.
OFS-ERROR-024	Failed to set ByteArray property of a RunForm object.
OFS-ERROR-025	Failed to set StringArray property of a RunForm object.
OFS-ERROR-027	Failed to process ByteArray property of a RunForm object.
OFS-ERROR-028	Failed to process StringArray property of a RunForm object.
OFS-ERROR-029	Failed to process nested message.
OFS-ERROR-030	Failed to do server-side connection.
OFS-ERROR-031	Failed to disconnect server-side connection.
OFS-ERROR-032	Failed because the server sent this error message: <message>
OFS-ERROR-033	Failed to expand the GUI control array.
OFS-ERROR-034	Failed to get the stored properties.
OFS-ERROR-035	Failed to send terminal message using server-side connection.
OFS-ERROR-036	Failed to send Forms message using server-side connection.
OFS-ERROR-037	Failed to process the Void property of a Button object.
OFS-ERROR-038	Failed to add listbox value. Bounds error. Listbox index: <index >, Listbox value: <value>
OFS-ERROR-039	Failed to find Listbox value. Value: <value>
OFS-ERROR-040	Failed to set the String property of an ErrorDialog object.
OFS-ERROR-041	Failed to process the String property of an ErrorDialog object.
OFS-ERROR-042	Failed to process the nested message of an ErrorDialog object.
OFS-ERROR-043	Failed to set the Point property of a FormWindow object.
OFS-ERROR-044	Failed to process the Point property of a FormWindow object.
OFS-ERROR-045	Failed to set the Boolean property of a FormWindow object.
OFS-ERROR-046	Failed to process the Boolean property of a FormWindow object.
OFS-ERROR-047	Failed to set the Integer property of a FormWindow object.
OFS-ERROR-048	Failed to process the Integer property of a FormWindow object.

OFS-ERROR-049	Failed to process the nested message of a FormWindow object.
OFS-ERROR-050	Failed to set the String property of a JavaContainer object.
OFS-ERROR-051	Failed to process the String property of a JavaContainer object.
OFS-ERROR-052	Failed to process the nested message of a JavaContainer object.
OFS-ERROR-053	Failed to set the String property of an LOV object.
OFS-ERROR-054	Failed to process the String property of an LOV object.
OFS-ERROR-055	Failed to set the Integer property of an LOV object.
OFS-ERROR-056	Failed to process the Integer property of an LOV object.
OFS-ERROR-057	Failed to set the Point property of an LOV object.
OFS-ERROR-058	Failed to process the Point property of an LOV object.
OFS-ERROR-059	Failed to set the Void property of an LOV object.
OFS-ERROR-060	Failed to process the Void property of an LOV object.
OFS-ERROR-061	Failed to process the nested message of an LOV object.
OFS-ERROR-062	Failed to set the String property of a LogonDialog object.
OFS-ERROR-063	Failed to process the String property of a LogonDialog object.
OFS-ERROR-064	Failed to process the nested message of a LogonDialog object.
OFS-ERROR-065	Failed to set the String property of a MenuParamDialog object.
OFS-ERROR-066	Failed to process the String property of a MenuParamDialog object.
OFS-ERROR-067	Failed to process the nested message of a MenuParamDialog object.
OFS-ERROR-068	Failed to set the String property of a PopList object.
OFS-ERROR-069	Failed to process the String property of a PopList object.
OFS-ERROR-070	Failed to set the Integer property of a PopList object.
OFS-ERROR-071	Failed to process the Integer property of a PopList object.
OFS-ERROR-072	Failed to set the Void property of a PopList object.
OFS-ERROR-073	Failed to process the Void property of a PopList object.
OFS-ERROR-074	Failed to process the nested message of a PopList object.

OFS-ERROR-075	Failed to set the String property of a TextField object.
OFS-ERROR-076	Failed to process the String property of a TextField object.
OFS-ERROR-077	Failed to set the Point property of a TextField object.
OFS-ERROR-078	Failed to process the Point property of a TextField object.
OFS-ERROR-079	Failed to set the Integer property of a TextField object.
OFS-ERROR-080	Failed to process the Integer property of a TextField object.
OFS-ERROR-081	Failed to set the Void property of a TextField object.
OFS-ERROR-082	Failed to process the Void property of a TextField object.
OFS-ERROR-083	Failed to process the nested message of a TextField object.
OFS-ERROR-084	Failed to set the Integer property of a Tree object.
OFS-ERROR-085	Failed to process the Integer property of a Tree object.
OFS-ERROR-086	Failed to set the String property of a Tree object.
OFS-ERROR-087	Failed to process the String property of a Tree object.
OFS-ERROR-088	Failed to process the nested message of a Tree object.
OFS-ERROR-089	Failed to process the nested message of a Button object.
OFS-ERROR-090	Failed to execute SSL handshake.
OFS-ERROR-091	Failed to collect Forms message.
OFS-ERROR-092	Failed to get the content length of the POST request.
OFS-ERROR-093	Failed to get new connection for a POST request.
OFS-ERROR-094	Failed to set up a new connection for an SSL-enabled POST request.
OFS-ERROR-095	Failed while posting a NULL request for a large-data response.
OFS-ERROR-096	Failed to store data from the server reply.
OFS-ERROR-097	Failed to do a re-POST request.
OFS-ERROR-098	Server reply data is invalid. If the following Java msg is null, server has terminated this Forms session. Java Msg: <message>
OFS-ERROR-099	Failed to disconnect the URL connection.
OFS-ERROR-100	Failed to connect to Forms Servlet. Check the URL and its parameters.

OFS-ERROR-101	Failed to do SSL connection to the Forms Servlet. Check the URL and its parameters.
OFS-ERROR-102	Failed to log the Forms Servlet connection.
OFS-ERROR-103	Failed to log the HTTP reply header.
OFS-ERROR-104	Failed while reading the http reply header. Server has terminated the Forms session.
OFS-ERROR-105	Failed while reading the server reply in an SSL connection.
OFS-ERROR-106	Failed to connect to the Forms Listener Servlet. Check the URL. If the URL is valid, the server is not accepting new connections.
OFS-ERROR-107	Failed to create a new URL connection for a Get request.
OFS-ERROR-108	Failed to connect to the Forms Listener Servlet in SSL mode. Check the URL. If the URL is valid, the server is not accepting new connections.
OFS-ERROR-109	Failed to log Listener Servlet connection.
OFS-ERROR-110	Failed to log initial Forms Server connection.
OFS-ERROR-111	Failed to get a URL connection for the first POST request.
OFS-ERROR-112	Server did not return the encryption keys for the first Post request. Forms Server is not accepting new connections.
OFS-ERROR-113	Failed to load loadplayer.java at startup. Library name: <libraryName>
OFS-ERROR-114	Failed to do SSL handshake.
OFS-ERROR-115	Failed to get SSL inputStream.
OFS-ERROR-117	Failed to close SSL socket.
OFS-ERROR-118	Failed to write the Forms Message.
OFS-ERROR-119	Failed to do a socket connection to the Forms Server.
OFS-ERROR-120	Server did not return the Forms encryption key during a socket connection.
OFS-ERROR-121	Failed to close the socket during a socket connection.
OFS-ERROR-122	Failed to write Forms message during a socket connection.
OFS-ERROR-123	Failed to send Forms message. Server terminated socket connection.
OFS-ERROR-124	Server reply is invalid. If Java msg is null, server has terminated this Forms session. Java Msg: <message>
OFS-ERROR-125	Failed to get the reply content. Check the URL. The URL may be invalid.

OFS-ERROR-126	JVM memory issues.
OFS-ERROR-128	LoadValue failed. LoadValue count is greater than the array length.

Heartbeat message failure on a virtual user

When a Player machine crashes or experiences a loss of communication, the heartbeat message that the Conductor sends out (if enabled) fails. This situation is indicated in the runtime Conductor through a message on each virtual user that is affected. When the heartbeat message fails for a virtual user, the Status column of the Details view of a script displays the following message: "The Player running this user failed to respond to a heartbeat message."

The option for enabling a heartbeat message is located on the [Player tab of the Options dialog box](#) in the Conductor.

Player

Overview of the QALoad Player

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

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

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

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

About the Player

Overview of the QALoad Player

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

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

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

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

QALoad Player menus

The following menus are available from the QALoad Player:

- File menu
- Edit menu
- View menu

[Options menu](#)
[Help menu](#)

Installing UNIX Players

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

You can access this guide by clicking

Start>Programs>Compuware>QALoad>Documentation>Installation and Configuration Guide.

Tuning QALoad Player for use with Oracle

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

- ! Set the Executing Threads Startup Interval parameter on the Player Configuration dialog box's Startup Parameters tab to between 2,000 and 4,000 milliseconds.
- ! Unless your application continually logs in and out of Oracle, move the logon commands (DO_olog and its associated DO_ologof) outside the Begin_Transaction/End_Transaction loop, where the Oracnvr program places them by default.

Dialog box and field descriptions

QALoad Player Main Window

The QALoad Player Main Window is divided into two parts:

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

- ! The bottom portion of the Player Main Window displays Player messages while a script is running.

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

Fields and Buttons

Compiled Script

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

Users

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

Transactions

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

Start

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

Abort

Click the Abort button to stop all virtual users immediately.

Exit

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

Debug Data

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

RR_Printf

Select this check box to display all RR_Printf commands contained in the script in the Player window.

RR_FailedMsg

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

Check Points

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

Auto Clear

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

Abort on Error

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

Create Timing File

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

Create JAR File

Select this check box so that when running a JAVA script, a JAR File will be created which contains all the dependencies of the script.

Run As

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

Save As

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

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

Player configuration

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

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

Runtime tab

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

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

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

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

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

Java tab

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

How to...

Installing UNIX Players


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

You can access this guide by clicking

Start>Program s>Compuware>QALoad>Documentation>Installation and Configuration Guide.

Transferring scripts to a UNIX Player

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

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

Transferring a Script

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

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

Start QALoad Player from the command line

QALoad Player may be started from a console using the following command line format:

```
Player [player name] [number of users] [TCP service port] [rebase]
```

Note that all parameters are optional and, if omitted, will default to the startup parameters as defined in the QALOAD.INI file.


Validate a script

To validate a script, follow these steps:

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

Setting up for DB2 playback

As with all QALoad middleware support on UNIX, DB2 UNIX support is replay only. QALoad does not support recording scripts from a UNIX environment. QALoad assumes that the DB2 environment is working prior to installation of QALoad.

 **Note:** To run DB2 load tests on AIX with 10 or more virtual users in thread-based mode, you must set the DB2 environment variable `EXTSHM` to `ON` to work around a memory handling problem in DB2.

To use EXTSHM with DB2:

1. Before starting the client application, type the following command:
`export EXTSHM=ON`
2. When starting the DB2 server, type the following commands:
`export EXTSHM=ON`
`db2set DB2ENVLIST=EXTSHM`
`db2start`

For information about setting up your UNIX Player installation, refer to the QACenter Performance Edition Installation and Configuration Guide. You can access this guide by clicking [Start>Programs>Compuware>QALoad >Documentation>Installation and Configuration Guide](#).

Analyze

Overview of QALoad Analyze

QALoad Analyze is the QALoad component used to create summary statistics and graphs from timing data collected during a load test. Set criteria for collecting and displaying test data in QALoad Analyze before or after opening a test's timing file (.tim). For example, alter output options, time ranges, and graphics display options.

Each time a timing file opens, QALoad Analyze automatically displays a Summary report in the Data window. In addition, QALoad Analyze generates a working folder where all files and reports related to the timing file are stored. QALoad Analyze provides seven [pre-defined reports](#) as well as the ability to create custom reports using XML file (.xml), XSL translation file (.xsl), and HTM file (.htm) formats. View these reports in QALoad Analyze or in a Web browser.

QALoad Analyze displays a timing file tab in the [Workspace](#), each tab containing groups. Use QALoad Analyze's interactive view to sort test data, produce detailed checkpoint data, produce a variety of graphs and reports (with drag and drop functionality), export data to different formats, and email test results and pre-defined reports.

About Analyze

Overview of QALoad Analyze

QALoad Analyze is the QALoad component used to create summary statistics and graphs from timing data collected during a load test. Set criteria for collecting and displaying test data in QALoad Analyze before or after opening a test's timing file (.tim). For example, alter output options, time ranges, and graphics display options.

Each time a timing file opens, QALoad Analyze automatically displays a Summary report in the Data window. In addition, QALoad Analyze generates a working folder where all files and reports related to the timing file are stored. QALoad Analyze provides seven [pre-defined reports](#) as well as the ability to create custom reports using XML file (.xml), XSL translation file (.xsl), and HTM file (.htm) formats. View these reports in QALoad Analyze or in a Web browser.

QALoad Analyze displays a timing file tab in the [Workspace](#), each tab containing groups. Use QALoad Analyze's interactive view to sort test data, produce detailed checkpoint data, produce a variety of graphs and reports (with drag and drop functionality), export data to different formats, and email test results and pre-defined reports.

Understanding durations

When you begin to analyze your test results, it is important to understand how durations are calculated by QALoad.

Transaction duration

Transaction duration is the time that the script being tested takes to complete a transaction, from the `BEGIN_TRANSACTION` command to the `END_TRANSACTION` command.

Three factors comprise transaction duration:

- ! The script processing time including, but not limited to, added script logic, QALoad processing of server replies, and other QALoad processing.

- ! Sleep time.
- ! The response time of the application under test including, but not limited to, the application server, database access, and network.

Checkpoint duration

Checkpoint duration is the amount of time between begin and end checkpoint statements. The following factors comprise checkpoint duration and apply to both automatic checkpoints and user-defined checkpoints

If you select the Conductor's Enable timing of automatic middleware checkpoints option or use the `BeginCheckpoint` and `EndCheckpoint` functions in the script, the following factors comprise checkpoint duration:

- ! The response time of the application under test, including, but not limited to, the application server, database access, and network.
- ! Sleep time, if the Conductor's **Include sleep times when calculating checkpoint timings** option is selected.
- ! QALoad processing time is not included within these checkpoints in order to provide a more accurate value of server, database, and network response times.

Checkpoint durations do not always sum to the same value as the transaction duration. For more information, see [Comparing checkpoint durations to transaction duration](#).

QALoad Analyze menus

QALoad Analyze menus and toolbar buttons

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

- File
- Edit
- View
- Tools
- Window
- Help

- Analyze Toolbar buttons
- Graph Toolbar buttons

File menu

Item	Description
Open	Opens a dialog box to select a timing file for Analyze to process. When a timing file is opened, Analyze opens a Summary report in the Data window and corresponding timing file tab in the Workspace.
Close	Closes the current timing file view displayed in the Data window. In the Workspace, the associated group and timing file tab will remain open.
Close Timing File	Closes the current timing file and all reports, detail views, or graphs associated with that timing file.
Print	Opens the Print dialog box to set options for printing the current Summary report.

Print Preview	Opens a window displaying the current Summary report as it will look when printed.
Print Setup	Opens the Print Setup dialog box to select a printer and to set options for printing.
Export	Opens the Save As dialog box to save a load test Detail view as a CSV (*.csv) file, HTML (*.htm, *.html) file, or save graphs to HTML (*.htm, *.html) files.
Send	Opens the Send dialog box, which enables you to save test data outside Analyze, package test data into .zip files, or email test data.
Properties	Opens the Properties dialog box, which displays details about Analyze and the current timing file. This dialog box also contains buttons that enable you to save the displayed information to the clipboard or in a file.
Exit	Exits the Analyze program.

Edit menu commands

Option	Description
Copy	Copies selected text to the clipboard.
Select All	Selects all counters, checkpoints, data points, etc., of the active timing file in the Workspace.
Unselect All	Un-selects all previously selected counters, checkpoints, data points, etc., in the Workspace of the active timing file.

View menu commands

Option	Description
Toolbar	Displays or removes the toolbar from the top of the screen.
Status Bar	Displays or removes the status bar from the bottom of the screen.
Detail	Opens a Detail information view in the Data Window for the selected checkpoints.

Graph	Opens the Select Graph dialog box to choose a graphical format for reviewing the current data.
as Web Page	Opens the current report, graph, or detail view, in the default Web browser.
Workspace	Displays or hides the Workspace.
Workbook	Displays or hides the Data window.

Tools menu commands

Option	Description
Options	Provides access to options for customizing data.
Show Replay	Launches the Conductor to replay a test recording .
Conductor	Starts QALoad's Conductor program, which is used to control all testing activity.
Workbench	Starts the QALoad Script Development Workbench, which is used to create and manage test scripts.

Window menu commands

Option	Description
Cascade	Automatically moves and resizes all the active windows, so they overlap one another.
Tile Horizontally	Moves and resizes all the active windows so they are lined up horizontally.
Tile Vertically	Moves and resizes all the active windows so they are lined up vertically.
Arrange Icons	Arranges any minimized windows within QALoad Analyze's parent window.
Close All	Closes all open windows.

Help menu

Option	Description
Help Topics	Displays QALoad's online help contents.
About QALoad Analyze	Displays the program's About box and copyright notice.

QALoad Analyze toolbars

Analyze toolbar buttons

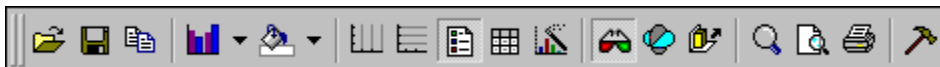
Click a toolbar button for a description of that button.



Graph toolbar buttons

Change the style and appearance of a graph using options available from the Graph toolbar. The Graph toolbar also contains buttons for standard Windows operations. Although it normally appears atop a graph, the toolbar is completely dockable. Move the toolbar to another side of the graph, or off the graph altogether, by clicking any unpopulated area of the toolbar and dragging it to another area.

Click any button in the following toolbar for a description of that button.



Display the Graph toolbar by right-clicking in an open area of a graph and choosing the Toolbar option from the shortcut menu.

Accessing test data

Using timing files

When you run a test using a particular session ID file (set up in the Conductor), each Player compiles a local timing file comprised of a series of timing records for each checkpoint of each script run on that Player. Each timing record in the file consists of a response time/elapsed time pair of values specifying the amount of time it took a certain checkpoint to finish (response time) at a specific time in the test (elapsed time).

At the end of a test, Player timing files are sent to the Conductor and are merged into a single timing file, called the Primary timing file, for analysis. If you set up integration with Compuware's ServerVantage product, the Conductor collects timing data from the ServerVantage central console and merges that data into the timing file as well.

Primary timing files are saved in the `\Program Files\Compuware\QALoad\TimingFiles` directory, and are named `<sessionID>_date_time.tim`.

The Primary timing file created by the Conductor after a test run contains all of the timing records of all Players in that test run. Use QALoad Analyze to view, sort, graph, and create reports using the test data in the timing file.

Hint: In the event that something goes wrong on the network and a Player timing file is not passed to the Conductor, it is still possible to analyze results from a Player timing file. Player timing files are saved in the

\Program Files\Compuware\QALoad\TimingFiles directory and are named **tim_yyyymmdd_hhmmss_xxx.ptf**, where **yyymmdd_hhmmss** is the date/time the test was started, and **xxx** is the Player number.

Accessing test data

When you open a timing file, QALoad's Analyze program summarizes the checkpoints recorded in the file during the load test and presents the data in a report format called the Summary report.

Three ways to access QALoad Analyze and open a timing file containing test results are:

To access Analyze from the QALoad Conductor:

1. In the QALoad Conductor, click **Tools>Options**. The Options dialog box appears.
2. Click the **General** tab. In the General Options area, select the **Launch Analyze After Test** check box.

At the end of each test run, QALoad Conductor automatically launches QALoad Analyze and opens the most recent timing file. Or, if you did not select the Launch Analyze After Test check box before the test:

1. Click **Tools>Analyze**.
2. In QALoad Analyze, click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.

Use the following method when accessing a previously-created timing file.

To access Analyze from the Windows Start menu:

1. Click **Start>Program Files>Compuware> QALoad >Analyze**.
2. Click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.

Use the following method when you are already working in the QALoad Script Development Workbench and need to access a previously-created timing file.

To access Analyze from the QALoad Script Development Workbench:

1. In the QALoad Script Development Workbench, click **Tools>Analyze**.
2. In QALoad Analyze, click **File>Open**. The Open Timing File dialog box appears. Select a timing file to work with by double-clicking the file name in the list of available timing files.

Accessing test data via groups

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

Access groups to select data for generating a detail view or graph. Click a group name below to view the type of data that is displayed by each group.

Reports
Checkpoints
Counters
Server Monitoring
Player Performance Counters

Top Processes
RIP Files

Displaying detail data


Displaying detail data

Display detailed statistics from a timing file such as checkpoints, counters, etc., in the QALoad Analyze Data window. View statistics for not only the active timing file, but also for other timing files and drag and drop onto the active timing file detail view.

To display detailed statistics:

1. In the workspace, with the appropriate Timing File tab selected, click the **group** for which you want to view statistics.
2. Select the appropriate checkpoints or counters (depending on which group you choose).
3. From the Analyze toolbar, click the **Detail** button or right-click on a selected checkpoint or counter and choose **Detail**.

Detail information is presented in the Data window in both a summary and data table. The information displayed varies based on the group selected.

 Note: If the test aborts, complete data for all the checkpoints and counters may not display.

The following detail views are available:

Checkpoints detail data

Checkpoint detail data is displayed in two panes: a summary table and a data table.

Checkpoints Summary Table

Shows statistical averages for the selected checkpoints and displays a summary of the raw data collected from the load test. The following data may be displayed:

Timing File: Name of the timing file the checkpoint came from.

Script: Name of the script file.

Checkpoint: Checkpoint name

Type: Type of checkpoint: response time or sleep time.

Group: Automatic (available if automatic checkpoint timings are enabled in the Conductor), or User (for user-defined checkpoints).

#Trans: Total number of data points that were used to calculate the statistics. If data thinning is enabled, this column displays as "#Thinned Trans".

#Recs: Number of records recording during the test. If data thinning is enabled, this column displays as "#Thinned Recs".

Data Thin: If the Enable Timing Data Thinning check box was selected in the QALoad Conductor's Timing Options dialog box prior to starting a load test, the value typed in the Thin Every <xx> Transactions will be in this column. If not selected, the value is none.

 Note: For a complete description of this QALoad Conductor option, see [Timing Options](#).

Min: Minimum recorded response time.

QALoad 5.02

Mean: Average of the response times.

Max: Maximum recorded response time.

StdDev: Standard deviation of all response times. Standard deviation is an indicator of how widely values are dispersed from the average (mean) value. A large standard deviation indicates a wide variance in response times.

Median: Median response time (in seconds). The median is the value at which half of the responses are greater and half of the responses are less. If the number of responses is large, the median is usually close to the mean.

Nth Percentile: Displays that nth% of the responses have a value less than the value shown.

Pacing (Seconds): Rate at which the script executed transactions.

VU's: Number of virtual users executing this script.

Checkpoints Data Table

Provides a view of the raw data collected during the load test. It can be useful for pinpointing anomalies with load test results. The following data may be displayed:

Timing File: Name of the timing file the checkpoint came from.

Script: Name of the script file.

Checkpoint: Checkpoint name.

Type: Type of checkpoint: response time or sleep time.

Group: Automatic (available if automatic checkpoint timings are enabled in the Conductor), or User (for user-defined checkpoints).

VU: The virtual user that was running.

Player: Player machine the test results came from.

#Samples: Displays how many records were thinned into a single record if data thinning was enabled. If the value is 1, the data records were not thinned.

Elapsed (Seconds): Time into the test at which a data point was collected.

Response (Seconds): Value of the data collected.

Counters detail data

Counter detail data is displayed as follows:


Counters Summary Table

Shows statistical averages for the selected counters. It is a summary of raw data collected from a load test. The following data may be displayed:


Timing File: Name of the timing file the counter came from.

Script: Name of the script file that contained the counter.

Group: Group name.

 **Note:** Not all counters will belong to a group. Those counters that do will have a group name displayed. For instance, custom and Web counters are logically organized by groups. However, Virtual Users and Total Virtual Users do not belong to a group.

Name: Name of the counter.

 **Note:** Certain statistical data is shared across all detail views. For a description of these fields, click the following: [Statistical Information](#)

Counters Data Table

Provides a view of the raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Script: Name of the script file that contained the counter.

Group: Group name. Custom and Web counters are logically organized by groups.

Name: Name of the counter.

Virtual User: Name of the virtual user.

Elapsed (Seconds): Time into the test at which a data point was collected.

Value: Value of the data collected. This column displays the value of instance counters.

Cumulative Value: Total number of occurrences during the elapsed time. This column displays the value of cumulative counters.

Server monitoring detail data

QALoad provides performance counter data through three server monitoring methods. Click the following links for descriptions of the server monitoring detail data options:

- ! [EcoTOOLS 6](#) - Availability Management application complimentary to QALoad for service level monitoring on UNIX of performance counters for applications, servers, and databases during production.
- ! [Remote Monitoring](#) - Monitoring of performance counters from a machine under test without the use of agent software on the machine.
- ! [Server Analysis](#) - Monitoring of performance counters from a machine under test using the ServerVantage agent software installed on the machine.
- ! [ServerVantage](#) - Availability Management application complimentary to QALoad for service level monitoring of performance counters for applications, servers, and databases during production. ServerVantage also provides notification, event management, and reporting features.

Player performance counters detail data

Player performance counters detail data is displayed as follows:

Player Performance Summary Table

Shows statistical averages for the selected Player performance counters. It is a summary of the raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Player: Player name.

Description: Description of the counter.

 **Note:** Certain [statistical data](#) is shared across all detail views.

Player Performance Data Table

Provides a view of the raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the performance counter came from.

Player: Player name.

Description: Name of the counter.

Elapsed (Seconds): Time into the test at which a data point was collected.

Value (%): Value of the data collected.

Top processes detail data

Top Processes detail data is displayed as follows:


Top Processes Summary Table

Shows statistical averages for the selected Top Processes data. It is a summary of the raw data collected from a load test. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Process: Name of the process monitored during the test.

#DataPts: Number of data points collected for each particular process.

 **Note:** Certain statistical data is shared across all detail views. For a description of these fields, click the following: Statistical Information

Top Processes Data Table

Provides a view of raw data collected during a load test. It can be useful for pinpointing anomalies within load test results. The following data may be displayed:

Timing File: Name of the timing file the counter came from.

Process: Process name.

Elapsed (Seconds): Time into the test at which a data point was collected.

% CPU: Percent of CPU used.

Sorting test data

A Detail view potentially contains a large number of checkpoints, counters, etc., especially if a load test had many virtual users. To make information manageable, specify up to three levels of criteria to sort by, in ascending or descending order.

For example, if a test ran using five scripts on 100 virtual users, sort the data by script name. Suppose each virtual user ran more than one transaction using a particular script, then sort by both script name and by virtual user. Or, to quickly locate any timing bottlenecks, sort by response time.

Use the Sort Details dialog box to sort a detail view. To access this dialog box, select Tools>Sort from the Analyze menu or click Sort on the Analyze toolbar.

Specify sort options for the following grid: Select the Summary option to conduct sort on the Summary table. Select the Data option to conduct a sort on the Data table.

Sort By: Select the first column to sort by from the list, then select the Ascending or Descending sort order option.

Then By: Select the second column to sort by from the list, then select the Ascending or Descending sort order option.

Then By: Select the third column to sort by from the list, then select the Ascending or Descending sort order option.

Creating a chart or graph

Thinning data before graphing

Test results may contain more data than can reasonably be graphed. Thinning data before graphing provides a more clear and manageable graph.

To set up data thinning:

1. With a timing file open, click **Tools>Options**.
2. Click the **Data Thinning** tab.
3. Type the number of data points to plot on each graph and select the method by which to graph the data points.
4. Click **OK**.

For a description of the options on this dialog box, see [Options Dialog Box - Data Thinning Tab](#).


Graphing QALoad timing data

A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph.

[Details](#)

Select the group to graph:

In the Workspace, with the appropriate Timing File tab selected, click the **group** for which to create a graph.

 **Note:** If the test aborts, complete data may not be available for all checkpoints and counters.

The following groups are available, depending on the timing file:


[Checkpoints](#)

[Counters](#)


[Server Monitoring](#)

[Player Performance Counters](#)

[Top Processes](#)

 **Note:** For each Group except Checkpoints, the graph type is a line graph. For graphing multiple checkpoints, the graph type is either a line or bar graph. For graphing a single checkpoint only, in addition to line and bar graphs, you can also create Response Time Distribution and Cumulative Response Time Distribution graphs.

Graphing checkpoints


 **Note:** A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph.

[Details](#)

To graph checkpoints:

1. Open the appropriate .tim file in QALoad Analyze. In the Workspace, click the Checkpoints group. Checkpoint data is listed in a tree-view.
2. Select the checkpoints to graph.
3. From the **View** menu, choose **Graph**. The Select Graph dialog box appears.


4. In the Graph Type drop-down list, select from the following:
 - **Line** (response times versus elapsed times for the selected data.)
 - **Bar** (median, mean, or a percentile response time of the selected checkpoints.)

 **Note:** The following graph types are only available when graphing a single checkpoint:

- **Response Time Distribution** (how the response times of a single checkpoint are distributed.)
- **Cumulative Response Time Distribution** (the percentage of checkpoint timings that were equal to or less than a specified value.)

Data for the selected checkpoint(s) is graphed in the Data window in the format selected in step 4.


Graphing counters

 **Note:** A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph.
[Details](#)

To graph counters:

1. Open the appropriate .tim file in QALoad Analyze. In the Workspace, click the **Counters** group. Counter data is listed in a tree-view.
2. Select the counter(s) to graph.
3. From the **View** menu, choose **Graph**. Data for the selected counter(s) is graphed in a line graph format in the Data window.

Graphing Player performance counters

 **Note:** A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph.
[Details](#)

To graph Player performance counters:

1. Open the appropriate .tim file in QALoad Analyze. Select the **Player Performance Counters** group.
2. In the Workspace, select the performance counter(s) to graph.
3. Click the **View Graph** button or right-click and choose **Graph** from the context menu. Data for the selected Agent(s) is graphed in a line graph format in the Data window.


Graphing server monitoring data

Monitoring servers is a method of load testing. QALoad provides performance counter data through three server monitoring methods:

- ! **EcoTOOLS 6** - An Availability Management application complementary to QALoad for service level monitoring of performance counters for applications, servers, and databases on UNIX during production.
- ! **Remote Monitoring** - Performs the monitoring of performance counters from a machine under test without the use of agent software on the machine.
- ! **Server Analysis** - Performs the monitoring of performance counters from a machine under test using the ServerVantage agent software installed on the machine.


- ! [ServerVantage](#) - An Availability Management application complementary to QALoad for service level monitoring of performance counters for applications, servers, and databases during production. ServerVantage also provides notification, event management, and reporting features.

Graphing top processes

 **Note:** A timing file can potentially contain enough data that graphing all of it at one time results in an unreadable graph. Before beginning, consider thinning the amount of data to be shown on a single graph. [Details](#)

To graph top processes:

1. Open the appropriate .tim file in QALoad Analyze. Select the **Top Processes** group.

 **Note:** The Top Processes group is available only if you enable the option in the QALoad Conductor Server Analysis Agent configuration screen before running a test.

2. In the Workspace, select the [data point\(s\)](#) to graph.
3. Click the **View Graph** button or right-click and choose **Graph** from the context menu. Data for the selected Agent(s) is graphed in a line graph format in the Data window.

Customizing a chart or graph

Customizing a graph

Change the style and appearance of a graph using options available from the [Graph toolbar](#). The Graph toolbar contains buttons for standard Windows operations as well as for customizing a graph's appearance. Display the Graph toolbar by right-clicking in an open area of a graph and choosing the [Toolbar](#) option from the shortcut menu. Although it initially appears above the graph, the toolbar is completely dockable. Move the toolbar to another side of the graph, or off the graph altogether, by clicking on any unpopulated area of the toolbar and dragging it to another area.

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

[Graph type](#)

[Color](#)

[Grid orientation](#) (horizontal and vertical)

[Legend box](#)

[Dimension](#) (3D or 2D)

[Rotation](#)

[Z-Cluster](#)

[Font](#)


Viewing reports

Viewing pre-defined reports

Pre-defined reports

QALoad Analyze provides pre-defined reports for viewing load test results without time-consuming data manipulation.

In the Workspace, select the Reports [group](#) and click the appropriate report. The reports are in HTML generated by XSL files. View them in QALoad Analyze, or [directly in a Web browser](#).

 **Note:** Compuware provides each of the available pre-defined reports as convenience to view the results of a load test without any data manipulation. In addition, create customized versions of these reports by selecting the appropriate group and creating [detail reports](#) and [graphs](#).

The following reports are available -- click a report name for details.

- [Summary](#)
- [Session](#)
- [Concurrent Users](#)
- [Response Time Analysis](#)
- [Output](#)
- [Client Throughput](#)
- [Server Monitoring](#)
- [Transaction Throughput](#)
- [Top Ten Longest Checkpoint Durations](#)
- [Player Performance](#)

Summary report

The Summary report is the primary output from each test run, one of the pre-defined reports QALoad Analyze makes available. When you open a timing file, QALoad Analyze automatically displays the Load Test Summary in the Data window. It presents timing information for each transaction in the timing file and the minimum, maximum, and median response times for each checkpoint. The output is divided into two sections. The first section presents the Summary Test Information and Test Time information. The second section presents the Script Information timing summaries for each script.

Sample Summary

For a brief description of each report section, scroll down and click a section heading, for example, Test Information, in the following sample.

Summary - perfect.tim									
Test Information									
Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages					
4	1	851	76	1,104					
Test Time									
Start	End	Duration	Pre	User	Post				
11/21/2003 - 12:48:57	11/21/2003 - 12:51:18	2 Minutes and 21 Seconds	00:00:00	00:02:21	00:00:00				
Data Thinning and Time Range									
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users				
1024 datapoint(s), Max	Max	90%	Seconds	No	851				
Script Information									
CP01									
Summary									
	Total	Thinned							
Peak Virtual Users	266	266							
Transaction Pacing	1 Second	1 Second							
Transaction Rate	43.46 Transactions Per Second	16.01 Transactions Per Second							
# Errors	74	74							
# Messages	6,167	1,024							
Counter Data Thinning	Disabled	Disabled							
Checkpoint Data Thinning	Disabled	Disabled							
Include Sleep Time	Yes	Yes							
Checkpoints									
ID	Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
0	Transaction Duration	2,048	1,024	3.0640	3.1816	3.5050	0.1177	3.1390	3.2440
1	Checkpoint: 1-1s	2,048	1,024	1.0010	1.0026	1.0320	0.0036	1.0020	1.0020
2	Checkpoint: 2-1s	2,048	1,024	1.0010	1.0018	1.0120	0.0020	1.0010	1.0020
3	Checkpoint: 3-1s	2,048	1,024	1.0010	1.0044	1.0310	0.0050	1.0020	1.0120

Session report

Provides summary information about the test session. The information in this report was obtained from the Conductor's configuration settings when the load test was started. To view a summary of test settings that includes changes made while the test was running, see the [Summary report](#).

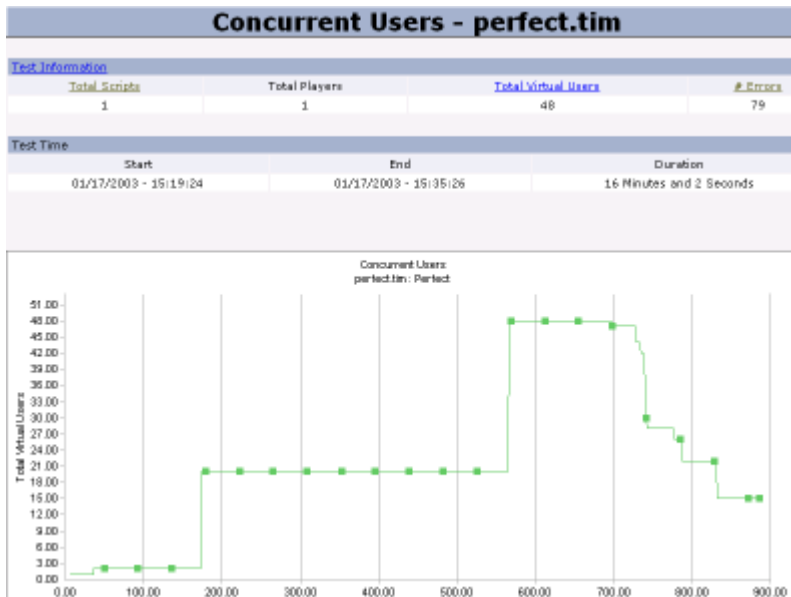
For descriptions of the information provided in each section, click the sections in the following image.

Session - perfect.tim									
Test Information									
Session	Perfect_Test1.ID								
Total Scripts	1								
Total Players	1								
Total Virtual Users	500								
Total Running Virtual Users	0								
Script Assignments									
Name	Middleware	Transactions	Auto Timings	Thin Every	Sleep %	Pacing	Threshold	On Error	
Perfect	WWW	0	True	1	Random	00:00:01.000	00:00:00	Restart	
Machines In Use									
Hostname	OS	RAM	Processor						
sprite	Windows 2000 Server Service Pack 2	256 MB	Intel Pentium III						
Machine Assignments									
Script	Start VUs	VU Increment	Interval	End VUs	Machine	Mode			
Perfect	1	0	00:00:00	500	sprite	Thread			

Concurrent Users report

Displays the total number of virtual users for the test, concurrent users vs. elapsed time, as well as graphs for individual scripts that were part of the test.

 **Note:** A totals graph will not display if the test contains only one script.



Response Time Analysis report

Provides an indicator of how well a script ran. The report displays a graph of each script's **transaction duration** (response time vs. elapsed time) as well as the following checkpoint summary data:

#Trans: Number of data points used to calculate the statistics.

#Recs: Number of data records. This value, if different from the value of #Trans, reflects the number of checkpoint records that are used for analysis after data thinning has been applied.

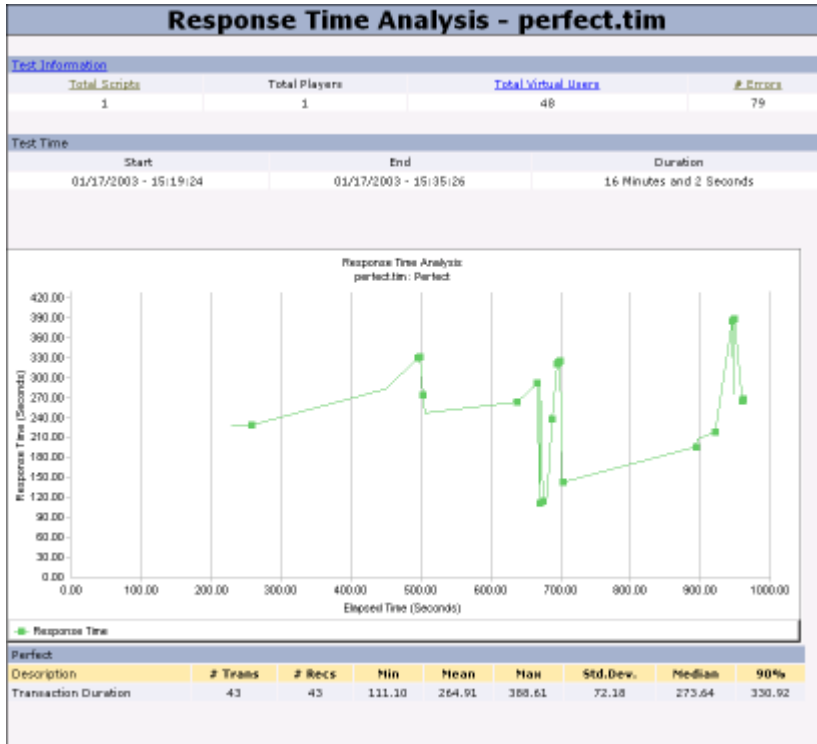
Min: Minimum recorded response time.

Max: Maximum recorded response time.

Std. Dev: Standard deviation of all response times. A large standard deviation indicates a wide variance in response times.

Median: Median response time, in seconds.

nth%: n percent of the responses have a value less than the value shown.



Output report

Provides a cumulative list of all errors, sorted by script and occurrence in time, that occurred during the course of a load test.

Note: Failed messages are included in the errors count that appears in the Test Information section of the report, but are detailed in the Script Messages section.



Client Throughput report

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



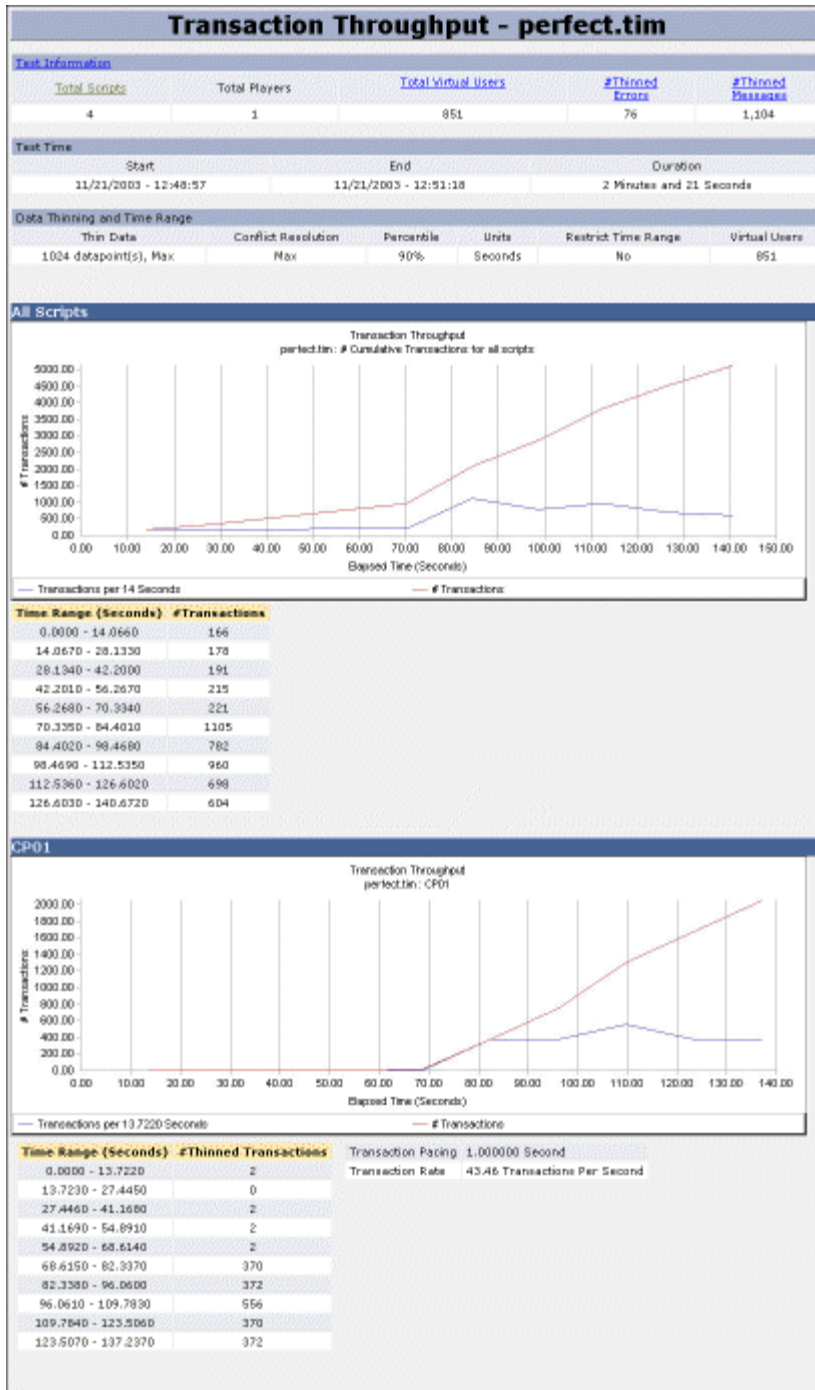
Server Monitoring report

Server monitoring is a component of load testing. QALoad provides performance counter data through three server monitoring methods: Remote Monitoring, ServerVantage, and Server Analysis Agent monitoring.



Transaction Throughput report

Provides the cumulative number of transactions over elapsed time for each script and for the total test.



Top Ten Longest Checkpoint Durations report

Provides graphs and lists details about checkpoints that had the longest checkpoint duration during the test. Checkpoints with longest durations are those that consumed the most amount of time during the test. This report contains the following sections:

- ! A summary section with overview information about the test.
- ! A bar graph of the ten longest checkpoint durations in the test, followed by details for each checkpoint in the graph. These checkpoints can originate in any script that was included in the test.
- ! Bar graphs for each script that show up to the ten longest checkpoint durations, followed by details for each checkpoint in the script.

QALoad 5.02

The report is generated by Analyze only if each script has at least one checkpoint other than the duration checkpoint. The data provided in the report can be used as a starting point to identify performance problems.

 **Note:** Transaction duration checkpoints are not included in the report.

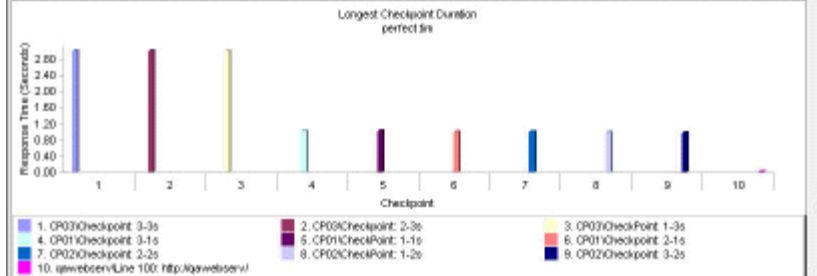
Top Ten Longest Checkpoint Durations - perfect.tim

Test Information				
Total Scripts	Total Players	Total Virtual Users	#Thinned Errors	#Thinned Messages
4	1	851	76	1,104

Test Time		
Start	End	Duration
11/21/2003 - 12:48:57	11/21/2003 - 12:51:18	2 Minutes and 21 Seconds

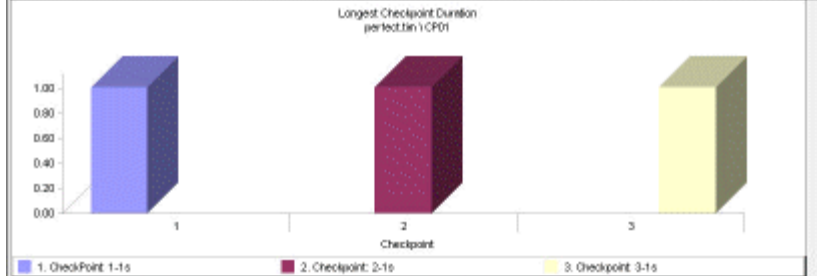
Data Thinning and Time Range					
Thin Data	Conflict Resolution	Percentile	Units	Restrict Time Range	Virtual Users
1024 datapoint(s), Max	Max	90%	Seconds	No	851

Longest Checkpoint Duration Summary



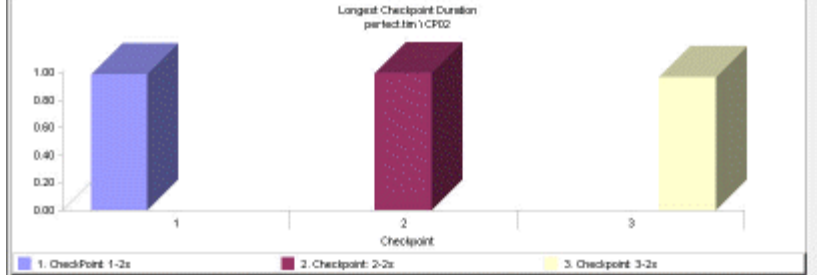
Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1. CP03/Checkpoint: 3-3s	1,024	75	3.0040	3.0054	3.0240	0.0035	3.0040	3.0050
2. CP03/Checkpoint: 2-3s	1,024	74	3.0040	3.0048	3.0240	0.0026	3.0040	3.0050
3. CP03/Checkpoint: 1-3s	1,024	82	3.0040	3.0046	3.0150	0.0020	3.0040	3.0050
4. CP01/Checkpoint: 3-1s	2,048	62	1.0010	1.0062	1.0310	0.0070	1.0020	1.0120
5. CP01/Checkpoint: 1-1s	2,048	62	1.0010	1.0046	1.0320	0.0060	1.0020	1.0120
6. CP01/Checkpoint: 2-1s	2,048	68	1.0010	1.0032	1.0120	0.0039	1.0020	1.0110
7. CP02/Checkpoint: 2-2s	1,024	819	0.0100	0.9924	2.0030	0.5679	1.0020	1.7650
8. CP02/Checkpoint: 1-2s	1,024	839	0.0000	0.9883	2.0030	0.5760	0.9810	1.7830
9. CP02/Checkpoint: 3-2s	1,024	832	0.0100	0.9662	2.0030	0.5956	0.9220	1.8130
10. qawebsevr\Line 100: http://qawebsevr/	1,024	1,024	0.0000	0.0027	0.0300	0.0045	0.0000	0.0100

CP01



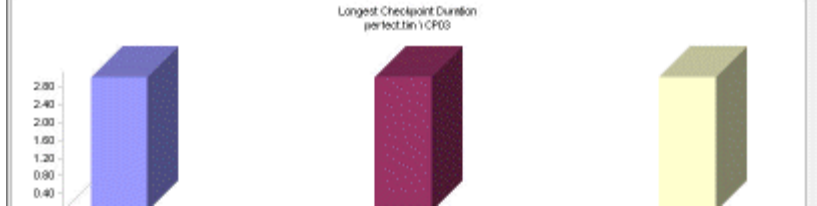
Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1. CheckPoint: 1-1s	2,048	62	1.0010	1.0046	1.0320	0.0060	1.0020	1.0120
2. Checkpoint: 2-1s	2,048	68	1.0010	1.0032	1.0120	0.0039	1.0020	1.0110
3. Checkpoint: 3-1s	2,048	82	1.0010	1.0062	1.0310	0.0070	1.0020	1.0120

CP02



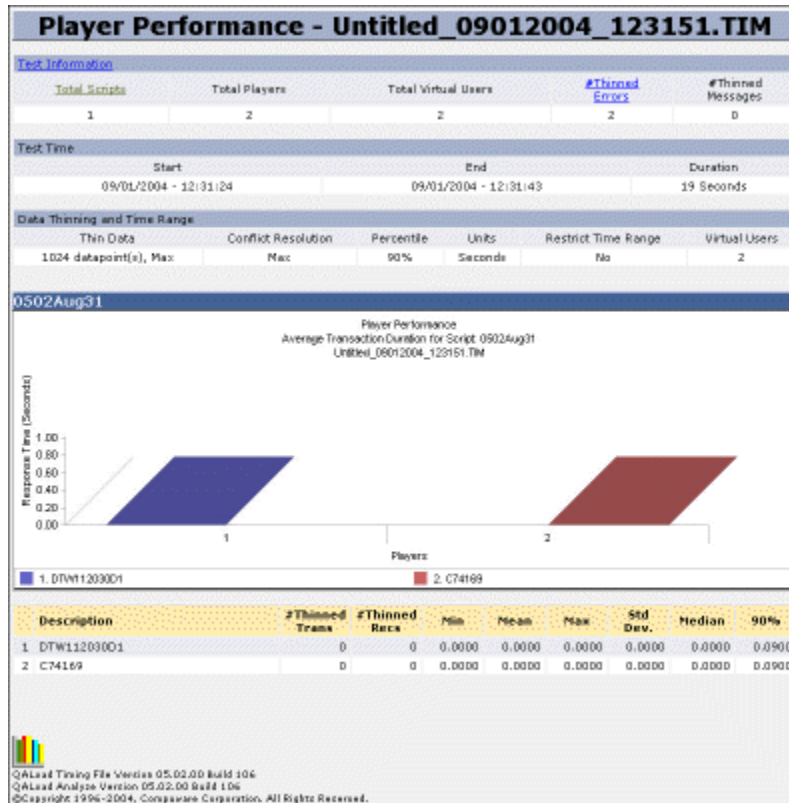
Description	#Thinned Trans	#Thinned Recs	Min	Mean	Max	Std Dev.	Median	90%
1. CheckPoint: 1-2s	1,024	839	0.0000	0.9883	2.0030	0.5760	0.9810	1.7830
2. Checkpoint: 2-2s	1,024	819	0.0100	0.9924	2.0030	0.5679	1.0020	1.7650
3. Checkpoint: 3-2s	1,024	832	0.0100	0.9662	2.0030	0.5956	0.9220	1.8130

CP03



Player Performance report

Displays transaction durations in a graph format by player machine. This report helps identify individual player machines that have poor test results. In addition to the bar graph that plots the average transaction duration for each player machine, the report also includes summary data for the overall test, and details for each player machine. This report is generated by Analyze only if two or more player machines were used in the test.



Viewing integrated reports

Application Expert and QALoad integrated reports

QALoad integrates with Application Expert version 8.0 and 9.0 to help analyze network performance during a load test. Application Expert is a Windows-based tool that enables users to examine the effects the network will have on the performance of new or modified applications prior to live deployment. Application Expert provides reports that help network managers identify poorly performing applications.

When using the Application Expert integration in a load test, QALoad generates a trace file. This file is the capture file created by the Application Vantage Agent.

Note: The trace file extension for Application Expert version 8.0 is .opt; for version 9.0 it is .opx.

The name of the trace file is <Session>_<YYYYMMDD>_<HHMMSS>.opt/.opx where <Session> is the name of the QALoad Conductor session used to execute the load test, and <YYYYMMDD> and <HHMMSS> are the date and time the trace file was captured. It is located in the (default) directory \ Program Files\ Compuware\ QALoad\ LogFiles.


Note: To generate the trace file, the ApplicationVantage Agent must be installed on the same machine as QALoad Conductor. The Agent can be installed during the QALoad installation or installed independently.

At the end of a load test, a high-level static report and various supporting files are automatically generated from the trace file and located in the directory \ Program

Files\ Compuware\ QALoad\ LogFiles\ <Session>_<YYYYMMDD>_<HHMMSS>. View the static report at any time in a Web browser such as Microsoft Internet Explorer. It contains the following Vantage views:

- ! Performance Overview
- ! Network Utilization and Transit Time
- ! Node Processing Detail
- ! Node Sending Detail
- ! Bounce Diagram
- ! Error Analysis
- ! Thread Analysis
- ! Conversion Map

A description of each view is provided in the <Session>_<YYYYMMDD>_<HHMMSS>.xml file.

 **Note:** To generate the high-level static report, Application Expert or Application Vantage must be installed on the same machine as QALoad Conductor.

For additional test analysis, import the trace file into Application Expert or Application Vantage. To use Application Expert or Application Vantage to further analyze the trace, refer to the Application Expert or Application Vantage User's Guides or online help.

Publishing or sharing test results

Exporting test data

Convert test data into three convenient formats for viewing or exporting:

HTML — Export data in a detail view or graph to HTML files for convenient viewing in a default Web browser or for sending as attachments in an email message. See [Exporting data to HTML](#) for instructions.

Comma-separated value (CSV) — Export data in a detail view to comma-separated value (CSV) files which can be imported into popular spreadsheet applications. See [Exporting data to CSV](#) for instructions.

RIP — Any time a user fails during load testing, QALoad Analyze generates a RIP file containing user errors. If a timing file has RIP file data, you can export the RIP file to the working folder and view it in QALoad Analyze or the QALoad Script Development Workbench. See [Exporting RIP file data](#) for instructions.

Exporting data to HTML

To export data from a detail view to HTML:


1. Open a timing file.
2. Generate a detail view or graph.
3. Click anywhere in the detail view or graph, making it active.
4. From the **File** menu, choose **Export>Data**. The Save As dialog box appears.
5. Navigate to the appropriate location for saving the HTML file and name the file.
6. Select **Web Page (*.htm;*.html)** as the file type and click **Save**.

Exporting data to CSV

To export data from the Detail view to a CSV file (*.csv):

1. Open a timing file.
2. Generate a detail view.
3. Click anywhere in the detail view making it active.
4. From the **File** menu, choose **Export>Data**. The Save As dialog box appears.
5. Navigate to the appropriate location for saving the file. In the **File name** field, type a name for the file.
6. Select **CSV (comma delimited) (*.csv)** as the file type and click **Save**.

Exporting RIP file data

 **Note:** If a timing file does not contain any RIP data, then a RIP Files group will not exist in the Workspace.

To export the RIP file data to the working folder:

1. Open a timing file.
2. In the Workspace, click the RIP Files group.
3. In the tree view, select the appropriate RIP files check box.
4. Right-click on the selected files and choose **Export**. The Browse For Folder dialog box appears.
5. Select the folder you wish to export the RIP file data to. The default is the [working folder](#).
6. Click **OK**. Analyze exports the RIP file to the working folder.

Sending email messages with test data

If you are using a Microsoft mail program, QALoad Analyze can send an email message with a timing file or pre-defined report attached. The recipient(s) of the message will be able to open the files in a Web browser.

To email pre-defined reports:

1. Choose **File>Send**.
2. In the **Send** dialog box, select reports, views, and timing files from their respective tabs and click **Add** to add them to the list of items you want to send.
3. In the **Send To** field, choose **Email Recipient**.
4. (optional) Click the **Zip to file** check box to send the files in the compressed .zip format. Type a name for the .zip file in the adjacent field.
5. Click **OK**. Analyze creates a new Outlook email message that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file as attachments, or a single .zip file that contains those files as an attachment. Address the email, add message text, and send the message.

Creating a .zip file of test results

You can create a .zip file to conveniently package all test data into one file for sending to others or storing locally. Analyze creates a file in .zip format, which you can either save to a location on your computer or send as an attachment to an email.

To create a .zip file:

1. Choose **File>Send**.
2. In the **Send** dialog box, select reports, views, and timing files from their respective tabs and click **Add** to add them to the list of items you want to include in the .zip file.
3. In the **Send To** field, choose **Email Recipient** to email the zip file or choose **File** to save the file on your computer.
4. Click the **Zip to file** check box to send the files in the compressed .zip format. Type a name for the .zip file in the adjacent field.
5. If you chose **File** in step 3, type the path of the location for the .zip file or click the browse button [...] to select a location.
6. Click **OK**. Depending on which option you chose in step 3, Analyze performs one of the following actions:
 - If you chose **Email Recipient**, Analyze creates a new Outlook email message that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file as a single, compressed .zip file attachment. Address the email, add message text, and send the message.
 - If you chose **File**, Analyze creates a single, compressed .zip file in the location you specified in step 5 that contains all of the pre-defined reports, .xml, .xsl, and files associated with the timing file.

Viewing reports

View reports generated by QALoad Analyze on a machine with QALoad installed or on any machine with a Web browser. In order to save the contents of a timing file's working folder, when viewing reports, clear the **Remove XML Working Folder** option. To properly set this option, see the **Workspace** tab on the **Options** dialog box. For more information, see [Options Dialog Box - Workspace Tab](#).

Viewing reports on a machine with QALoad Analyze

To view reports in QALoad Analyze, click the **Summary** report button or any of the pre-defined report buttons in the QALoad Analyze Workspace. See [Load Test Summary](#) for a quick introduction to viewing reports.

Viewing reports on a machine without QALoad Analyze

To view reports in a Web browser, copy the entire working folder for the timing file onto the machine. The following files are required (where <Summary> represents the name of the report):

- ! <Summary>.htm
- ! <Summary>.xml
- ! <Summary>.xsl

In addition, the Microsoft XML version 4.0 parser (provided with QALoad) is required to view QALoad reports. View any of the pre-defined reports by clicking the <Summary>.htm file to launch a report with the assistance of the associated XML and XSL support files.

Other ways to view test data

View not only pre-defined reports, but also timing file detail views and graphs by exporting or sending email messages with test data to another machine. Click the following links for more information:

- ! [Exporting Test Data](#)
- ! [Sending Email Messages with Test Data](#)

Viewing test results in a Web browser

An important part of the load testing process is viewing and studying the results of a test. You can view the results of a load test not only on a machine where QALoad is installed, but also on any machine with a Web browser. QALoad Analyze provides pre-defined reports as well as .xml and .xsl files which can be customized to meet desired specifications.

When you open a timing file, QALoad Analyze generates a working folder containing all supporting files, reports, and images generated from that timing file. This folder is located in the directory `\Program Files\Compuware\QALoad\TimingFiles\<xxx>.xml.source` where `<xxx>` is the name of the timing file.

The following files are found in the working folder:

File Name	Description
<code><timingfile>.xml.source</code>	Working folder generated in the Reports folder when opening a timing file. The working folder name is always the <code><name of the timing file></code> with a <code>.xml.source</code> extension.
<code><timingfile>.xml</code>	Original timing file with just enough information to create the QALoad Analyze pre-defined reports. It is a representation of the timing file, <code><timingfile>.tim</code> .
<code><timingfile>.complete.xml</code>	Original timing file containing all data collected during a load test. It can be an extremely large file. Use this file if creating a report using XSL that required this data.
<code>summary.htm</code>	Use this HTML file to view the Summary report (or any other available pre-defined report) in any Web browser.
<code>summary.xml</code>	Generated XML file for the Summary report (or any other available pre-defined report.)
<code>summary.xsl</code>	Generated XSL file for the Summary report (or any other available pre-defined report.) Translates the .xml file specifying HTML as its output and generates the HTML report. Use this file to customize the reports by writing in .xsl.
<code>default.htm</code>	Report which provides a main screen to launch any other pre-defined reports. Uses <code>nav.htm</code> for the navigation frame.

When closing a timing file, either keep all of the reports generated from the timing file in the working folder, or delete them. To set this option, see the [Workspace tab on the Options dialog box](#).

To view load test results in a Web browser, click: [How to View Reports](#).

Language Reference

Contents of QALoad Language Reference

The QALoad Language Reference provides command reference information for the following general and middleware-specific commands:

- ADO
- Citrix
- DB2
- ODBC
- ODBC/DB2
- Oracle 7
- Oracle 7/8
- Oracle 8
- Oracle Forms Server
- QALoad
- QARun Integration
- SAP Versions 4.x
- SAP Versions 6.20/6.40
- SSL
- Tuxedo
- Uniface
- Uniface Polyserver
- Winsock
- WWW

ADO

ADO Index

[ADO_Command\(n\)->Cancel](#)

Terminates the execution of an asynchronous method call.

[ADO_Command\(n\)->CreateParameter](#)

Creates a new Parameter object with a specified name, type, direction, size, and value. Any values passed in the arguments are written to the corresponding Parameter properties. This method does not automatically append the Parameter object to the Parameters collection of a Command object.

[ADO_Command\(n\)->Execute](#)

Executes the query specified in the CommandText property or CommandStream property of the object.

[ADO_Command\(n\)->GetCommandStream](#)

Retrieves the value contained in the CommandStream property of this instance of the ADO Command object. The CommandStream property is retrieved using a pointer to a variant.

[ADO_Command\(n\)->GetCommandText](#)

Retrieves the value of the CommandText property for this instance of the ADO Command object. A string is returned as its argument.

[ADO_Command\(n\)->GetCommandTimeout](#)

Retrieves the value contained within the CommandTimeout property of this instance of the ADO Command object.

[ADO_Command\(n\)->GetCommandType](#)

Retrieves the value contained in the CommandType property of the current instantiation of the Command object.

[ADO_Command\(n\)->GetDialect](#)

Retrieves the value contained within the Dialect property of this instance of the ADO Command object.

[ADO_Command\(n\)->GetName](#)

Allows the script to retrieve the value of the Name property for this instance of the ADO Command object.

[ADO_Command\(n\)->GetNamedParameters](#)

Retrieves the NamedParameters property of the Command object.

[ADO_Command\(n\)->GetParameters](#)

Retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

[ADO_Command\(n\)->GetPrepared](#)

Retrieves the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

[ADO_Command\(n\)->GetProperties](#)

Retrieves the complete set of properties for this particular instance of the Command object. PropertySets may change for different providers.

[ADO_Command\(n\)->PutActiveConnection](#)

Determines the Connection object affected by the specified Command object or ADO Recordset.

[ADO_Command\(n\)->PutCommandText](#)

Sets the value of the CommandText property for this instance of the ADO command object.

[ADO_Command\(n\)->PutCommandTimeout](#)

Sets the value contained within the CommandTimeout property of this instance of the ADO Command object.

[ADO_Command\(n\)->PutCommandType](#)

Sets the value for the CommandType property of the current instantiation of the Command object.

[ADO_Command\(n\)->PutDialect](#)

Sets the value of the Dialect property of this instance of the ADO Command object.

[ADO_Command\(n\)->PutName](#)

Enables the script to set the value of the Name property for this instance of the ADO Command object.

[ADO_Command\(n\)->PutNamedParameters](#)

Sets the value of the NamedParameters property of the command object.

[ADO_Command\(n\)->PutPrepared](#)

The PutPrepared method call sets the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

[ADO_Command\(n\)->PutRefActiveConnection](#)

Determines the ADO Connect object affected by the specified ADO Command object or ADO Recordset. Also sets the pointer to the QALoad ADO Connect object for this instance of the actual ADO Command object.

[ADO_Connect\(n\)->BeginTrans](#)

Begins a new transaction.

[ADO_Connect\(n\)->Close](#)

Closes a Connection object.

[ADO_Connect\(n\)->CommitTrans](#)

Save changes, ends the transaction. May start a new transaction.

ADO_Connect(n)->Execute

Executes the query passed in the CommandText argument on the connection to the method.

ADO_Connect(n)->GetAttributes

GetAttributes is read/write. It's value is the sum of one or more XactAttributeEnum values. The default is zero (0).

ADO_Connect(n)->GetCommandTimeout

Returns the value of the timeout in a pointer to a long.

ADO_Connect(n)->GetConnectionString

GetConnectionString method retrieves the value of ConnectionString property of this instantiation of the ADO Connect object. The ConnectionString specifies a data source by passing argument=value statements. These are separated by semi-colons.

ADO_Connect(n)->GetConnectionTimeout

Use GetConnectionTimeout on a Connection object to cancel a connection attempt, if necessary, due to network or server delays. If the time interval specified in the Connection Timeout property setting runs out before a connection can be opened, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

ADO_Connect(n)->GetCursorLocation

Lets you choose a cursor location from those accessible to the provider.

ADO_Connect(n)->GetDefaultDatabase

Retrieves the value of the DefaultDatabase property from this instance of the ADO Connect object.

ADO_Connect(n)->GetIsolationLevel

Sets a Connection object's isolation level. Takes effect the next time the BeginTrans method is called.

ADO_Connect(n)->GetMode

Sets or returns access permissions for the current connection. The GetMode property can only be set after the Connection object is closed.

ADO_Connect(n)->GetProvider

Returns the provider name for a connection.

ADO_Connect(n)->GetState

Determines the state of a specified object at any time.

ADO_Connect(n)->GetVersion

Returns the version of ADO that is being used.

ADO_Connect(n)->Open

Establishes the connection to the data source.

ADO_Connect(n)->OpenSchema

Returns information about the data source. For example, tables, columns included in the tables, data types, etc.

ADO_Connect(n)->PutAttributes

Sets the transaction attribute for this connection object.

ADO_Connect(n)->PutCommandTimeout

PutCommandTimeout Sends a timeout in seconds before the command will timeout with an error.

ADO_Connect(n)->PutConnectionString

Specifies a data source

ADO_Connect(n)->PutConnectionTimeout

Use PutConnectionTimeout on a Connection object to abandon an attempt to connect due to network or server delays. If a connection is not made in the specified time, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

[ADO_Connect\(n\)->PutCursorLocation](#)

Lets you choose a cursor library from those accessible to the provider.

[ADO_Connect\(n\)->PutDefaultDatabase](#)

Sets the default database within a connection object.

[ADO_Connect\(n\)->PutIsolationLevel](#)

Sets the isolation level of a Connection object.

[ADO_Connect\(n\)->PutMode](#)

Sets the access permissions being used on the current connection.

[ADO_Connect\(n\)->PutProvider](#)

Sets the provider name for a connection.

[ADO_Connect\(n\)->RollbackTrans](#)

Reverses changes made in an open transaction and ends the transaction. This is linked with BeginTrans. This will only be seen in the script if a transaction fails for some reason. If it fails and you see this call, look over the script logic and see if the transaction can be committed.

[ADO_Field\(n\)->AppendChunk](#)

A special data handling method that writes data, in chunks, to the Field object.

[ADO_Field\(n\)->GetActualSize](#)

Retrieves the value contained within the ActualSize property of this instance of the ADO Field object.

[ADO_Field\(n\)->GetAttributes](#)

Retrieves the value contained within the Attributes property of this instance of the ADO Field object.

[ADO_Field\(n\)->GetChunk](#)

Retrieves chunks of binary or character data to an appropriate buffer.

[ADO_Field\(n\)->GetDataFormat](#)

Retrieves an IUnknown instance describing the data format for this field.

[ADO_Field\(n\)->GetDefinedSize](#)

Retrieves the value contained within the DefinedSize property of this instance of the ADO Field object.

[ADO_Field\(n\)->GetName](#)

Retrieves the value contained within the Name property of this instance of the ADO Field object. Note that the actual ADO call has a BSTR as the argument; therefore, there is some data conversion occurring within this call.

[ADO_Field\(n\)->GetNumericScale](#)

Retrieves the value contained within the NumericScale property of this instance of the ADO Field object.

[ADO_Field\(n\)->GetOriginalValue](#)

Retrieves the value contained within the Value property of this instance of the ADO Field object before any changes were made permanent by a call to an update method.

[ADO_Field\(n\)->GetPrecision](#)

Retrieves the value contained within the Precision property of this instance of the ADO Field object.

[ADO_Field\(n\)->GetProperties](#)

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

[ADO_Field\(n\)->GetStatus](#)

Retrieves the value contained within the Status property of this instance of the ADO Field object. It takes a pointer to a long as an argument. Within this parameter, the Value of the status property of this instance of the Field object is returned.

[ADO_Field\(n\)->GetType](#)

Retrieves the value contained within the Type property of this instance of the ADO Field object. The Actual

ADO call uses another enumerated type, `DataTypeEnum`, to handle the datatypes. Conversion to this type happens within the QALoad call.

[ADO_Field\(n\)->GetUnderlyingValue](#)

Specifies the current value of a Field object in the database, after any updates to the recordset.

[ADO_Field\(n\)->GetValue](#)

Retrieves the value of a ADO Field object into a pointer to a Variant. The argument will handle any type of data. This can be done by using the Variant datatype. Data is retrieved into a pointer to a Variant.

[ADO_Field\(n\)->PutAttributes](#)

The `PutAttributes` method call sets the value contained within the `Attributes` property of this instance of the ADO Field object.

[ADO_Field\(n\)->PutDataFormat](#)

This updates the current value of the data format updates to the ADO Recordset.

[ADO_Field\(n\)->PutDefinedSize](#)

Sets the value contained within the `DefinedSize` property of this instance of the ADO Field object.

[ADO_Field\(n\)->PutNumericScale](#)

Sets the value contained within the `NumericScale` property of this instance of the ADO Field object.

[ADO_Field\(n\)->PutPrecision](#)

Sets the value contained within the `Precision` property of this instance of the ADO Field object.

[ADO_Field\(n\)->PutType](#)

Sets the value contained within the `Type` property of this instance of the ADO Field object.

[ADO_Field\(n\)->PutValue](#)

Resets the value of this instance of the ADO Field object. This is the first step in updating a Recordset's value.

[ADO_FieldSet\(0\)->GetNewEnum](#)

In order to iterate through each ADO Field in a ADO FieldSet collection, an `ADOIEnumField` object is returned. The `GetNewEnum` call on the ADO FieldSet object creates the ADO IEnumField object allowing the enumeration to take place.

[ADO_FieldSet\(n\)->Append](#)

`Append` creates and appends a new Field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprises a mechanism for updating or retrieving information from a Data Provider.

[ADO_FieldSet\(n\)->Append15](#)

`Append15` creates and appends a new field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprise a mechanism for updating or retrieving information from a Data Provider. The `Append15` function does NOT allow the user to add the data to this ADO Field object. It creates the ADO Field object in the ADO FieldSet collection, but does not add in the data.

[ADO_FieldSet\(n\)->CancelUpdate](#)

Cancels changes made to the current or new row of an ADO Recordset object, or the ADO Fieldset collection of an ADO Record object, before calling the `Update` method.

[ADO_FieldSet\(n\)->Delete](#)

Deletes an object from the Fields collection.

[ADO_FieldSet\(n\)->GetCount](#)

The method returns the number of ADO Field objects contained within the ADO FieldSet collection.

[ADO_FieldSet\(n\)->GetItem](#)

This call retrieves a ADO Field object from this instance of the ADO FieldSet collection. The result of the

call is that a ADO Field object is brought back to be manipulated within the script. ADO Field retrieval is a part of the variablization process.

[ADO_FieldSet\(n\)->Refresh](#)

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO FieldSet collection has no visible effect.

[ADO_FieldSet\(n\)->Resync](#)

Synchronizes the values of a Record object's Fields collection with the data source. The Count property is not affected by this method.

[ADO_FieldSet\(n\)->Update](#)

Saves any changes you make to the ADO FieldSet collection of a Record object.

[ADO_IEnum\(n\)->NextProperty](#)

Enumeration through collections of properties should be done very carefully, because in the example below, we will reset all of the properties to the same value. To reset different values, get rid of the loop and set each property individually.

[ADO_IEnumField\(n\)->NextField](#)

Enumeration through collections of ADO Fields should be done very carefully, because in the example given below, the script checks the status of each of the different ADO Fields. In order to do more meaningful work, reset values of different ADO Fields then break them out of the loop and use the PutValue call to place new values into the ADO Field objects.

[ADO_IEnumParameter\(n\)->NextParameter](#)

Enumeration through collections of ADO Parameters should be done very carefully, because in the example given below, the script checks different values of each of the different ADO Parameters. In order to do some more meaningful work, resetting values of different ADO Parameters then break them out of the loop and use the PutValue call to place new values into the ADO Parameter objects.

[ADO_LoadVariant\(n\)](#)

[ADO_Parameter\(n\)->AppendChunk](#)

A special data handling method that writes data, in chunks, to the Parameter object.

[ADO_Parameter\(n\)->GetAttributes](#)

Retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

[ADO_Parameter\(n\)->GetDirection](#)

Retrieves the value contained within the Direction property of this instance of the ADO Parameter object.

[ADO_Parameter\(n\)->GetName](#)

Retrieves the value contained within the Name property of this instance of the ADO Parameter object.

[ADO_Parameter\(n\)->GetNumericScale](#)

Retrieves the value contained within the NumericScale property of this instance of the ADO Parameter object. Returns a Byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

[ADO_Parameter\(n\)->GetPrecision](#)

Retrieves the value contained within the Precision property of this instance of the ADO Parameter object. Returns a Byte value showing the maximum number of digits used to represent values for a numeric Parameter object. The Precision property is read/write.

[ADO_Parameter\(n\)->GetSize](#)

Retrieves the value contained within the Size property of this instance of the ADO Field object.

[ADO_Parameter\(n\)->GetValue](#)

Use the Value property to return data from ADO Parameter objects and to return parameter values with ADO Parameter objects.

ADO_Parameter(n)->PutAttributes

This method call retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutDirection

Indicates Parameter type: input, output, input and output, or the return value from a stored procedure.

ADO_Parameter(n)->PutName

Sets the value contained within the Name property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutNumericScale

Sets the value contained within the NumericScale property of this instance of the ADO Parameter object. Sends a byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

ADO_Parameter(n)->PutPrecision

Sets the value contained within the Precision property of this instance of the ADO Parameter object. Sends a byte value showing the maximum number of digits used to represent values for a numeric ADO Parameter object. The Precision property is read/write.

ADO_Parameter(n)->PutSize

Retrieves the value contained within the Size property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutType

Sets the value contained within the Type property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutValue

Sets the value contained within the Value property of this instance of the ADO Parameter object.

ADO_ParameterSet(n)->Append

Appends a ADO Parameter object to the collection of ADO Parameters.

ADO_ParameterSet(n)->Delete

Deletes an ADO Parameter object from the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetCount

The method returns the number of ADO Parameter objects contained within the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetItem

Locates a specific ADO Parameter in the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetNewEnum

In order to iterate through all of the ADO Parameters in an ADO ParameterSet collection, an ADO IEnumParameter object is returned. The GetNewEnum call on the ADO ParameterSet object creates the ADO IEnumParameter object allowing the enumeration to take place.

ADO_ParameterSet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO ParameterSet collection has no visible effect.

ADO_Property(n)->GetAttributes

Describes column characteristics by setting or returning a Long value.

ADO_Property(n)->GetName

Retrieves the value of the Name attribute of this instance of the Property object.

ADO_Property(n)->GetType

Indicates a property's type as conveyed as a DataTypeEnum.

ADO_Property(n)->GetValue

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

[ADO_Property\(n\)->PutAttributes](#)

Describes column characteristics by returning a long value, which indicates characteristics of the table represented by the Column object. The value can be a combination of ColumnAttributesEnum constants. The default value is zero (0), which is neither adColFixed nor adColNullable.

[ADO_Property\(n\)->PutAttributes](#)

Describes column characteristics by returning a long value, which indicates characteristics of the table represented by the Column object. The value can be a combination of ColumnAttributesEnum constants. The default value is zero (0), which is neither adColFixed nor adColNullable.

[ADO_Property\(n\)->PutValue](#)

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

[ADO_PropertySet\(n\)->GetCount](#)

The method returns the number of ADO Property objects contained within the ADO PropertySet collection.

[ADO_PropertySet\(n\)->GetItem](#)

Retrieves a specific ADO Property in the ADO PropertySet collection.

[ADO_PropertySet\(n\)->GetNewEnum](#)

In order to iterate through all of the ADO Property objects in an ADO PropertySet collection, an ADOEnum object is returned. The GetNewEnum call on the ADO PropertySet object creates the ADO IEnum object allowing the enumeration to take place.

[ADO_PropertySet\(n\)->Refresh](#)

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO PropertySet collection has no visible effect.

[ADO_Record\(n\)->Cancel](#)

Cancels execution of a pending, asynchronous method call.

[ADO_Record\(n\)->Close](#)

Use to close a Recordset, Record, or Stream object. Any associated data or exclusive access you may have had to the data through this particular object will be released. You can reopen the object later using the Open method.

[ADO_Record\(n\)->CopyRecord](#)

Copies a file or directory (including its contents) to another location.

[ADO_Record\(n\)->DeleteRecord](#)

Deletes a file or directory, and all its subdirectories.

[ADO_Record\(n\)->GetActiveConnection](#)

Use the ActiveConnection property to determine the ADO Connect object over which the specified ADO Record object will execute.

[ADO_Record\(n\)->GetChildren](#)

Returns an ADO Recordset, in the form of a Pointer to an ADO Recordset object, whose rows represent the files and subdirectories in the directory represented by this Record.

[ADO_Record\(n\)->GetFields](#)

Contains all the Field objects of an ADO Recordset or ADO Record object.

[ADO_Record\(n\)->GetMode](#)

Sets or returns the access permissions being used on the current connection by the provider.

[ADO_Record\(n\)->GetParentURL](#)

Sets the current value of the source property for this instance of the actual ADO Command object.

[ADO_Record\(n\)->GetRecordType](#)

This method is used to check the contents of the ADO RecordType property for this instance of ADO Record object, returning the RecordTypeEnum in a pointer to a long.

[ADO_Record\(n\)->GetSource](#)

Indicates the entity represented by the ADO Record object.

[ADO_Record\(n\)->GetState](#)

You can use the State property to determine the state of a given ADO Record object at any time.

[ADO_Record\(n\)->MoveRecord](#)

Moves a file, or a directory and its contents, to another location.

[ADO_Record\(n\)->Open](#)

Makes the call through to the Open method within the ADO Record object to open an existing ADO Record object, or create a new file or directory.

[ADO_Record\(n\)->PutActiveConnection](#)

PutActiveConnection is read/write when the ADO Record object is closed. It may contain a connection string or reference to an open ADO Connect object. When the ADO Record object is open and contains a reference to an open ADO Connect object, PutActiveConnection is read-only.

[ADO_Record\(n\)->PutMode](#)

Sets the access permissions being used on the current connection by the provider. You can only set this property when the ADO Connect object is closed.

[ADO_Record\(n\)->PutRefActiveConnection](#)

Specifies the ADO Connect object to be affected by the specified ADO Record object.

[ADO_Record\(n\)->PutSource](#)

Sets the current value of the source property for this instance of the actual ADO Command object. The Source property must refer to an object existing within the scope of that ADO Connect.

[ADO_Recordset\(n\)->_xClone](#)

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it makes a clone of the calling ADO Recordset. This is given the arguments and the method name.

[ADO_Recordset\(n\)->_xResync](#)

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it resynchronizes the ADO Recordset with the underlying data provider. This is given the arguments and the method name.

[ADO_Recordset\(n\)->_xSave](#)

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it saves ADO Recordset data to the location given in the first argument. This is given the arguments and the method name.

[ADO_Recordset\(n\)->AddNew](#)

Creates a new record for an updatable ADO Recordset object.

[ADO_Recordset\(n\)->Cancel](#)

Cancels execution of a pending, asynchronous method call.

[ADO_Recordset\(n\)->CancelBatch](#)

Cancels any pending updates in an ADO Recordset that is in batch update mode.

[ADO_Recordset\(n\)->CancelUpdate](#)

Cancels any changes made to the current row or discards a new row of an ADO Recordset object before calling the Update method.

[ADO_Recordset\(n\)->Clone](#)

Duplicates an ADO Recordset object. Can specify that the clone be read-only.

[ADO_Recordset\(n\)->Close](#)

Closes an open object and any dependent objects.

[ADO_Recordset\(n\)->CompareBookmarks](#)

Compares two bookmarks. Returns an indication of their relative values.

[ADO_Recordset\(n\)->Delete](#)

Use to delete the current record or a group of records.

[ADO_Recordset\(n\)->Find](#)

Locates a row in an ADO Recordset that matches specified criteria.

[ADO_Recordset\(n\)->GetAbsolutePage](#)

Identifies, by page number, where the current record resides.

[ADO_Recordset\(n\)->GetAbsolutePosition](#)

Specifies the ordinal position of the current record of an ADO Recordset object.

[ADO_Recordset\(n\)->GetActiveCommand](#)

Specifies the ADO Command object which created an ADO Recordset object.

[ADO_Recordset\(n\)->GetActiveConnection](#)

For a Command, ADO Recordset, or ADO Record object, specifies the associated ADO Connect object.

[ADO_Recordset\(n\)->GetBOF](#)

Determines if an ADO Recordset object contains records or if you've gone beyond its limits while moving from record to record.

[ADO_Recordset\(n\)->GetBookmark](#)

Indicates a bookmark identifying an ADO Recordset object's current record, or sets the current record to that identified by a bookmark.

[ADO_Recordset\(n\)->GetCacheSize](#)

Specifies the number of records in the ADO Recordset that are cached locally.

[ADO_Recordset\(n\)->GetCollect](#)

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

[ADO_Recordset\(n\)->GetCursorLocation](#)

Specifies the location of the cursor service.

[ADO_Recordset\(n\)->GetCursorType](#)

Specifies the type of cursor to use when opening the ADO Recordset object.

[ADO_Recordset\(n\)->GetDataMember](#)

Specifies the data member to be retrieved from the object referenced by the DataSource property.

[ADO_Recordset\(n\)->GetDataSource](#)

Specifies an object containing data to be represented as an ADO Recordset object.

[ADO_Recordset\(n\)->GetEditMode](#)

Specifies the current record's editing status.

[ADO_Recordset\(n\)->GetEOF](#)

Indicates that the current record position is after the last record in an ADO Recordset object.

[ADO_Recordset\(n\)->GetFields](#)

Returns a container of an ADO Recordset or ADO Record object's Field objects.

[ADO_Recordset\(n\)->GetFilter](#)

Specifies a filter for data in an ADO Recordset.

[ADO_Recordset\(n\)->GetIndex](#)

This is a hidden method. It is undocumented within MSDN.

ADO_Recordset(n)->GetLockType

Specifies the type of locks placed on records during editing.

ADO_Recordset(n)->GetMarshalOptions

Specifies records to be marshaled back to the server.

ADO_Recordset(n)->GetMaxRecords

Specifies the maximum number of records to return to an ADO Recordset from a query.

ADO_Recordset(n)->GetPageCount

Specifies the number of pages of data contained in the ADO Recordset object.

ADO_Recordset(n)->GetPageSize

Indicates the number of records that make up a single page in the ADO Recordset.

ADO_Recordset(n)->GetProperties

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

ADO_Recordset(n)->GetRecordCount

Indicates the number of records in an ADO Recordset object.

ADO_Recordset(n)->GetRows

Retrieves multiple records of an ADO Recordset object into an array.

ADO_Recordset(n)->GetSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

ADO_Recordset(n)->GetSource

Indicates the data source for a Recordset object.

ADO_Recordset(n)->GetState

Indicates for all applicable objects whether the state of the object is open or closed.

ADO_Recordset(n)->GetStatus

Indicates the status of the current record with respect to batch updates or other bulk operations.

ADO_Recordset(n)->GetStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

ADO_Recordset(n)->GetString

Returns the ADO Recordset as a string.

ADO_Recordset(n)->Move

Moves the position of the current record in an ADO Recordset object.

ADO_Recordset(n)->MoveFirst

Use the MoveFirst method to move the current record position to the first record in the ADO Recordset.

ADO_Recordset(n)->MoveLast

Use the MoveLast method to move the current record position to the last record in the ADO Recordset. The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

ADO_Recordset(n)->MoveNext

Use the MoveNext method to move the current record position one record forward (toward the bottom of the ADO Recordset). If the last record is the current record and you call the MoveNext method, ADO sets the current record to the position after the last record in the ADO Recordset (EOF is True). An attempt to move forward when the EOF property is already True generates an error.

ADO_Recordset(n)->MovePrevious

Use the MovePrevious method to move the current record position one record backward (toward the top of

the ADO Recordset). The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the MovePrevious method, ADO sets the current record to the position before the first record in the ADO Recordset (BOF is True).

[ADO_Recordset\(n\)->NextRecordset](#)

Clears the current ADO Recordset object and returns the next ADO Recordset by advancing through a series of commands.

[ADO_Recordset\(n\)->Open](#)

Using the Open method on an ADO Recordset object opens a cursor that represents records from a base table, the results of a query, or a previously saved ADO Recordset.

[ADO_Recordset\(n\)->PutAbsolutePage](#)

Indicates on which page the current record resides.

[ADO_Recordset\(n\)->PutAbsolutePosition](#)

Indicates the ordinal position of an ADO Recordset object's current record.

[ADO_Recordset\(n\)->PutActiveConnection](#)

Indicates to which Connection object the specified Command, ADO Recordset, or Record object currently belongs.

[ADO_Recordset\(n\)->PutBookmark](#)

Indicates a bookmark that uniquely identifies the current record in an ADO Recordset object or sets the current record in an ADO Recordset object to the record identified by a valid bookmark.

[ADO_Recordset\(n\)->PutCacheSize](#)

Indicates the number of records in the ADO Recordset that are cached locally.

[ADO_Recordset\(n\)->PutCollect](#)

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit. Neither QALoad support professionals nor development recommend adding this method to a script.

[ADO_Recordset\(n\)->PutCursorLocation](#)

Indicates the location of the cursor service.

[ADO_Recordset\(n\)->PutCursorType](#)

Use the CursorType property to specify the type of cursor that should be used when opening the ADO Recordset object.

[ADO_Recordset\(n\)->PutDataMember](#)

Indicates the name of the data member that will be retrieved from the object referenced by the DataSource property.

[ADO_Recordset\(n\)->PutFilter](#)

Indicates a filter for data in an ADO Recordset.

[ADO_Recordset\(n\)->PutIndex](#)

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

[ADO_Recordset\(n\)->PutLockType](#)

Indicates the type of locks placed on records during editing.

[ADO_Recordset\(n\)->PutMarshalOptions](#)

Indicates which records are to be marshaled back to the server.

[ADO_Recordset\(n\)->PutMaxRecords](#)

Indicates the maximum number of records to return to an ADO Recordset from a query.

[ADO_Recordset\(n\)->PutPageSize](#)

Indicates how many records constitute one page in the ADO Recordset.

ADO_Recordset(n)->PutRefActiveConnection

Indicates to which ADO Connect object the specified ADO Command, ADO Recordset, or Record object currently belongs.

ADO_Recordset(n)->PutRefDataSource

Indicates an object that contains data to be represented as an ADO Recordset object.

ADO_Recordset(n)->PutRefSource

Sets a Command object as the data source for a Recordset object.

ADO_Recordset(n)->PutSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

ADO_Recordset(n)->PutSource

Indicates the data source for an ADO Recordset object.

ADO_Recordset(n)->PutStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

ADO_Recordset(n)->ReQuery

Updates the data in an ADO Recordset object by re-executing the query on which the object is based.

ADO_Recordset(n)->Resync

Refreshes the data in the current ADO Recordset object, or Fields collection of a Record object, from the underlying database.

ADO_Recordset(n)->Save

Saves the ADO Recordset in a file or ADO Stream object.

ADO_Recordset(n)->Seek

The SeekEnum is an Enumerated value giving the direction of the seek operation.

ADO_Recordset(n)->Supports

Determines whether a specified ADO Recordset object supports a particular type of functionality.

ADO_Recordset(n)->Update

Saves any changes you make to the current row of an ADO Recordset object.

ADO_Recordset(n)->UpdateBatch

Writes all pending batch updates within the ADO Recordset to disk.

ADO_Stream(n)->Cancel

Cancels execution of a pending ADO Stream, asynchronous method call.

ADO_Stream(n)->Close

Closes an open object and any dependent objects.

ADO_Stream(n)->CopyTo

Copies the specified number of characters or bytes (depending on Type) in the ADO Stream to another ADO Stream object.

ADO_Stream(n)->Flush

Forces the contents of the ADO Stream remaining in the ADO buffer to the underlying object with which the ADO Stream is associated.

ADO_Stream(n)->GetCharset

Indicates the character set into which the contents of a text ADO Stream should be translated.

ADO_Stream(n)->GetEOS

Indicates whether the current position is at the end of the ADO Stream.

ADO_Stream(n)->GetLineSeparator

Indicates the binary character to be used as the line separator in text ADO Stream objects.

[ADO_Stream\(n\)->GetMode](#)

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

[ADO_Stream\(n\)->GetPosition](#)

Indicates the current position within an ADO Stream object.

[ADO_Stream\(n\)->GetSize](#)

Returns a Long value that specifies the size of the ADO Stream in number of bytes. The default value is the size of the ADO Stream, or -1 if the size of the ADO Stream is not known.

[ADO_Stream\(n\)->GetState](#)

The ADO Stream object's State property can have a combination of values. For example, if a statement is executing, this property will have a combined value of adStateOpen and adStateExecuting.

[ADO_Stream\(n\)->GetType](#)

Indicates the type of data contained in the ADO Stream (binary or text).

[ADO_Stream\(n\)->LoadFromFile](#)

Loads the contents of an existing file into an ADO Stream.

[ADO_Stream\(n\)->PutCharset](#)

Indicates the character set into which the contents of a text ADO Stream should be translated.

[ADO_Stream\(n\)->PutLineSeparator](#)

Indicates the binary character to be used as the line separator in text ADO Stream objects.

[ADO_Stream\(n\)->PutMode](#)

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

[ADO_Stream\(n\)->PutPosition](#)

Indicates the current position within an ADO Stream object.

[ADO_Stream\(n\)->PutType](#)

Indicates the type of data contained in the ADO Stream (binary or text).

[ADO_Stream\(n\)->Read](#)

Reads a specified number of bytes from a binary ADO Stream object.

[ADO_Stream\(n\)->ReadText](#)

Reads specified number of characters from a text ADO Stream object.

[ADO_Stream\(n\)->SaveToFile](#)

Saves the number of bytes contents of the current ADO Stream to the file from the current position. It sends the second param number of bytes to that File.

[ADO_Stream\(n\)->SetEOS](#)

Sets the current position within the ADO Stream as the End of the ADO Stream

[ADO_Stream\(n\)->SkipLine](#)

Skips one entire line when reading a text ADO Stream.

[ADO_Stream\(n\)->Write](#)

Write writes BINARY Data to the ADO Stream buffer.

[ADO_Stream\(n\)->WriteText](#)

Writes a specified text string to an ADO Stream object.

[ADOStream\(n\)->Open](#)

Opens an ADO Stream object to manipulate streams of binary or text data.

[ExtractVariantValue](#)

Retrieves the contents of a variant and places that value in a string.

[PrintVariant](#)

Decodes variant data and places this data into a string.

Object definitions

Objects are an encapsulation of methods and data. Microsoft's ADO (ActiveX Data Objects) model encapsulates properties (data) and methods. A property contains the characteristics of a particular instance of an object. In order to determine the state of an object, you need to retrieve a property set on that object.

In order to give a brief overview of ADO and how QALoad works with ADO, this section contains definitions for each of the QALoad ADO Objects and an explanation of the QALoad ADO Container Object, the QALoad ADO Wrapper Object, the QALoad ADO Object Instance, and the Microsoft ADO Object.

The QALoad ADO Container Object is created and run by the script. It contains all of the corresponding QALoad ADO Wrapper Objects in the script. A container is essentially a collection of wrapper objects. The QALoad ADO Wrapper Object wraps the Microsoft ADO Objects. For example, if the application under test uses 15 recordsets, then the recorded QALoad script will have a QALoad CADORecordSet Container Object that will contain 15 QALoad CARecordSet Wrapper Objects which correspond to Microsoft ADO RecordSet Objects.

The QALoad ADO Object Instance is the instantiation of the wrapper objects used in the script to manipulate the Microsoft ADO Objects. The object instances are indexed in the script (counting upwards from zero.)

To see the relationship between each of the QALoad ADO Container Objects, their corresponding QALoad ADO Wrapper Objects, and the Microsoft ADO Objects they emulate, see the following table:

QALoad ADO Container Objects	QALoad ADO Wrapper Objects	QALoad ADO Object Instance	Microsoft ADO Objects
CADOCommand	CACommand	ADO_Command(n)	Command
CADOConnect	CAConnect	ADO_Connect(n)	Connection
CADOEnum	CAEnum	ADO_IEnum(n)	IEnum
CADOField	CAField	ADO_Field(n)	Field
CADOFieldSet	CAFieldSet	ADO_FieldSet(n)	Fields
CADOParameter	CAParameter	ADO_Parameter(n)	Parameter
CADOParameterSet	CAParameterSet	ADO_ParameterSet(n)	Parameters
CADOProperty	CAProperty	ADO_Property(n)	Property
CADOPropertySet	CAPropertySet	ADO_Propertyset(n)	Properties
CADORecord	CARecord	ADO_Record(n)	Record
CADORecordSet	CARecordSet	ADO_Recordset(n)	RecordSet
CADOStream	CAStream	ADO_Stream(n)	Stream

ADO_Command(n)->Cancel

Terminates the execution of an asynchronous method call.

Syntax

```
ADO_Command(n)->Cancel();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Command(0)->CreateParameter( "ParamTest1", adEmpty, adParamInput, 0, pvValue,
ADOParameter[0] );
ADOParameter.Release( 0 );
ADO_LoadVariant( pvValue, "8", "A Test Value" );
ADO_Command(0)->CreateParameter( "TestParam2", adEmpty, adParamInputOutput, 100, pvValue,
ADOParameter[0] );
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->CreateParameter

Creates a new Parameter object with a specified name, type, direction, size, and value. Any values passed in the arguments are written to the corresponding Parameter properties. This method does not automatically append the Parameter object to the Parameters collection of a Command object.

CreateParameter lets you set additional properties whose values ADO will validate when appending the Parameter object to the collection. If specifying a variable length data type in the Type argument, you must either pass a Size argument, or set the Size <mdprosize.htm> property of the Parameter object before appending it to the Parameters collection. Otherwise, an error occurs.

If you specify a numeric data type (adNumeric or adDecimal) in the Type argument, then you must also set the NumericScale <mdpronumericscale.htm> and Precision <mdproprecision.htm> properties.

Syntax

```
ADO_Command(n)->CreateParameter( char* sName, adTypeEnum Type, adDirectionEnum Direction,
long nLength, VARIANT* pvValue, CAPParameter* pParameter );
```

Parameters

Parameter	Description
n	An index to the object.
char* sName	String containing the name of the parameter.
adTypeEnum Type	Enumerated type describing the data type.
adDirectionEnum Direction	Enumerated type describing the direction of parameter: in, out, in/out, return, or unknown.
long nLength	An integer variable specifying the maximum length, in bytes, of the parameter.
VARIANT* pvValue	Value of parameter at time of creation.
CAPParameter* pParameter	Created parameter.

Example

```
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdUnknown );
ADO_Command(0)->GetState( pLong );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Command(0)->CreateParameter( "ParamTest1", adEmpty, adParaminput, 0, pvValue,
ADOParameter[0] );
ADO_LoadVariant( pvValue, "8", "A Test Value" );
```

ADO_Command(n)->Execute

Executes the query specified in the CommandText property or CommandStream property of the object.

Syntax

```
ADO_Command(n)->Execute( VARIANT* pvRecNo, VARIANT*
pvParamList, long nCommandType, CRecordset* pRecordset );
```

Parameters

Parameter	Description
n	An index to the object.
VARIANT* pvRecNo	A long variable to which the provider returns the number of records that the operation affected.
VARIANT* pvParamList	A variant array of parameter values passed.
long nCommandType	A long value that indicates how the provider should evaluate the CommandText property of the Command object.
CRecordset* pRecordset	QALoad wrapper object, returned ADORecordset.

Example

```
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
ADO_LoadVariant( pvValue, "3", "-1" );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Command(0)->Execute( pvValue, pvSource, 1,
ADORecordset[0] );
```

ADO_Command(n)->GetCommandStream

Retrieves the value contained in the CommandStream property of this instance of the ADO Command object. The CommandStream property is retrieved using a pointer to a variant.

Syntax

```
ADO_Command(n)->GetCommandStream( VARIANT* pvValue );
```

Parameters

Parameter	Description
n	An index to the object.

VARIANT* pvValue	A pointer to a variant into which the data is retrieved.
------------------	--

Example

```
ADO_Command(0)->GetCommandStream( pvValue );
ADO_Command(0)->GetDialect( pvSource );
```

ADO_Command(n)->GetCommandText

Retrieves the value of the CommandText property for this instance of the ADO Command object. A string is returned as its argument.

The CommandText property is returned inside a CLoadString.

Syntax

```
ADO_Command(n)->GetCommandText( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	The CommandText property for which a value is to be returned.

Example

```
ADO_Command(0)->PutCommandText("Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
ADO_Connect(0)>Open("PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa; pwd="" ;database=Master;" , "",
"", -1 );
```

ADO_Command(n)->GetCommandTimeout

Retrieves the value contained within the CommandTimeout property of this instance of the ADO Command object.

Use CommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Command(n)->GetCommandTimeout( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing the time interval.

Example

```
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 250 );
```

ADO_Command(n)->GetCommandType

Retrieves the value contained in the CommandType property of the current instantiation of the Command object.

GetCommandType should always be combined with adCmdText or adCmdStoredProc (for example, adCmdText+adExecuteNoRecords). Note that using adExecuteNoRecords with the Open method or a Command object used by that method returns an error.

Syntax

```
ADO_Command(n)->GetCommandType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing a command type Enum.

Example

```
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->GetDialect

Retrieves the value contained within the Dialect property of this instance of the ADO Command object. The Dialect property contains a valid GUID (Globally Unique Identifier) which represents the dialect of the command text or stream. The default value for this property is {C8B521FB-5CF3-11CE-ADE5-00AA0044773D}, indicating that the provider should choose how to interpret the command text or stream.

Syntax

```
ADO_Command(n)->GetDialect( CLoadString sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
CLoadString sLoadStr	A CLoadString. Value of the Dialect property for this instance of the ADO Command.

Example

```
ADO_Command(0)->PutName( "MyTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutDialect( "{C8B521FB-5CF3-11CE-ADE5-00AA0044773D}" );
ADO_Command(0)->GetDialect( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
```

ADO_Command(n)->GetName

Allows the script to retrieve the value of the Name property for this instance of the ADO Command object.

Syntax

```
ADO_Command(n)->GetName( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString. Value of the Name property for this instance of the ADO Command.

Example

```
ADOConnect.Release( 0 );
ADO_Command(0)->PutName( "aTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
```

ADO_Command(n)->GetNamedParameters

Retrieves the NamedParameters property of the Command object. When true, the Command object handles the parameters by name as opposed to order. The method of retrieving parameters depends on the value of the NamedParameters property.

Syntax

```
ADO_Command(n)->GetNamedParameters( VARIANT_BOOL* pnParameter );
```

Parameters

Parameter	Description
pnParameter	A short evaluating to either True (-1) or False (0).


Example

```
ADO_Command(0)->PutCommandTimeout( 250 );
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetNamedParameters( pVTBOOL );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->GetParameters

Retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

A Command object has a collection of Parameters associated with it, holding zero or more Parameters within it. Using the Refresh method on a Command object's Parameters collection retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

 **Note:** The *n* associated with the ADO_Command object and that associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
ADO_Command(n)->GetParameters( ADOParameterSet[n] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOParameterSet[n]	An instance of the ADO ParameterSet object. This instance is being created and filled as a result of this call.

Example

```
ADO_Command(0)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
ADO_Command(0)->PutCommandText( "op_Getparamvb6" );
BeginCheckpoint( "ADOCommand::GetParameters" );
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
EndCheckpoint( "ADOCommand::GetParameters" );
```

ADO_Command(n)->GetPrepared

Retrieves the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

A VARIANT_BOOL is a short evaluating to either True (-1) or False (0).

As a result, the provider will save a compiled version of the query specified in the CommandText property before a Command object's first execution. This might slow down the initial execution; however, performance will improve in subsequent executions because the provider will use the compiled version of the command.

Syntax

```
ADO_Command(n)->GetPrepared( pVTBOOL );
```

Parameters

Parameter	Description
n	An index to an object.
pVTBOOL	Pointer to the VARIANT_BOOL value contained in the Prepared property of this instance of the ADO Command object.

Example

```
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->GetProperties

Retrieves the complete set of properties for this particular instance of the Command object. PropertySets may change for different providers.

Syntax

```
ADO_Command(n)->GetProperties( CAPropertySet* pPropertySet );
```

Parameters


Parameter	Description
n	An index to the object.
CAPropertySet *nPropertySet	Set of CAProperty objects. Each CAProperty object contains a single characteristic, a piece of data, which partially describes the state of a particular instance of an object.

Example

```
ADO_LoadVariant( pvValue, "8", "test_number" );
ADO_Command(0)->GetItem( pvValue, ADOCommand[0] );
ADO_Command(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Command(n)->PutActiveConnection

Determines the Connection object affected by the specified Command object or ADO Recordset. Preceding each call to PutActiveConnection, a call to LoadVariant must be made. This flavor of LoadVariant takes ADO Connect information and loads that into a pointer to a variant.

 **Note:** The n associated with the ADO_Command object and that associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
LoadVariant ( pvValue, ADOConnect[n] );
ADO_Command(n)->PutActiveConnection( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	Pointer to a variant containing the ADO Connection Object.

Example

```
ADO_Command(2)->PutCommandText( "sp_GetParameterSet" );
ADO_Command(2)->PutCommandType( adCmdStoredProc );
LoadVariant( pvValue, ADOConnect[0] );
ADO_Command(2)->PutActiveConnection( pvValue );
BeginCheckpoint( "ADOCommand::GetParameters" );
ADO_Command(2)->GetParameters( ADOParameterSet[0] );
EndCheckpoint( "ADOCommand::GetParameters" );
```

ADO_Command(n)->PutCommandText

Sets the value of the CommandText property for this instance of the ADO command object. One method of setting up a command, for instance a SQL statement of some sort, would be by using PutCommandText.

Syntax

```
ADO_Command(n)->PutCommandText( "Text" );
```

Parameters

Parameter	Description
n	An index to the object.
text	The ANSI value of the CommandText property.

Example

```
ADO_Command(0)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
BeginCheckpoint( "ADOCommand::GetParameters" );
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
EndCheckpoint( "ADOCommand::GetParameters" );
```

ADO_Command(n)->PutCommandTimeout

Sets the value contained within the CommandTimeout property of this instance of the ADO Command object.

Use CommandTimeout on a Connection or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Command(n)->PutCommandTimeout( long nCommandTimeout );
```

Parameters

Parameter	Description
n	An index to the object.
long nCommandTimeout	Timeout value (a non-negative integer) to set for this instance of the ADO Command object.

Example

```
ADO_Connect(0)->PutConnectionString( "DSN=My;UID=sa; PWD=" " ;" );
BeginCheckpoint( "ADOConnect::Open" );
ADO_Connect(0)->Open( "", "", "", -1 );
EndCheckpoint( "ADOConnect::Open" );
ADO_Command(0)->PutCommandTimeout( 200 );
```

ADO_Command(n)->PutCommandType

Sets the value for the CommandType property of the current instantiation of the Command object.

This is generally done using an enumerated type — in this case the CommandTypeEnum would be used. Since all of the different enumerated types within ADO essentially boil down to longs, longs are accepted as the argument.



Caution: Randomly introducing numbers into the script to do this operation will have unpredictable results.

PutCommandType should always be combined with adCmdText or adCmdStoredProc (for example, adCmdText+adExecuteNoRecords). Using adExecuteNoRecords with the Open method, or a Command object used by that method, results in an error.

Syntax

```
ADO_Command(3)-> PutCommandType ( adCmdText );
```

Parameters

Parameter	Description
n	An index to the object.
CommandTypeEnum	The enumerated datatype identifies the type of Command.

Example

```
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->PutDialect

Sets the value of the Dialect property of this instance of the ADO Command object.

The Dialect property contains a valid GUID (Globally Unique Identifier) that represents the dialect of the command text or stream. The default value for this property is {C8B521FB-5CF3-11CE-ADE5-00AA0044773D}, which indicates that the provider should choose how to interpret the command text or stream.

When the Dialect property is set, ADO validates the GUID and raises an error if the value supplied is not a valid GUID. Refer to your provider documentation to determine the GUID values supported by the Dialect property.

Syntax

```
ADO_Command(n)->PutDialect( "GUID" );
```

Parameters

Parameter	Description
n	An index to the object.
"GUID"	A unique identifier for a dialect.

Example

```
ADO_Command(0)->PutName( "MyTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutDialect( "{C8B521FB-5CF3-11CE-ADE5-00AA0044773D}" );
ADO_Command(0)->GetDialect( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
```

ADO_Command(n)->PutName

Enables the script to set the value of the Name property for this instance of the ADO Command object.

Syntax

```
ADO_Command(n)->PutName( "Name" );
```

Parameters

Parameter	Description
n	An index to the object.
Name	Value of the Name property for this instance of the ADO Command object.

Example

```
ADOConnect.Release( 0 );
ADO_Command(0)->PutName( "aTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
```

ADO_Command(n)->PutNamedParameters

Sets the value of the NamedParameters property of the command object. When true, the Command object handles the parameters by name as opposed to order. The method of retrieving parameters depends on the value of the NamedParameters property.

Syntax

```
ADO_Command(n)->PutNamedParameters( short nParameter );
```

Parameters

Parameter	Description
short nParameter	A short evaluating to either True (-1) or False (0).

Example

```
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetNamedParameters( pVTBOOL );
ADO_Command(0)->PutNamedParameters( 0 );
```

ADO_Command(n)->PutPrepared

The PutPrepared method call sets the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

A VARIANT_BOOL is a short evaluating to either True (-1) or False (0).

Syntax

```
ADO_Command(n)->PutPrepared( -1 );
```

Parameters


Parameter	Description
n	An index to the object.
flag	TRUE (-1) or FALSE (0).

Example

```
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->PutRefActiveConnection

Determines the ADO Connect object affected by the specified ADO Command object or ADO Recordset. Also sets the pointer to the QALoad ADO Connect object for this instance of the actual ADO Command object.

 **Note:** The n associated with the ADO Command object and that associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
ADO_Command(n)->PutRefActiveConnection( ADOConnect[n] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOConnect[n]	The connection object (ADO Connect).

Example

```
ADO_Command(1)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(1)->PutCommandType( adCmdStoredProc );
ADO_Command(1)->PutCommandText( "op_Getparamvb6_Batch" );
BeginCheckpoint("ADOCommand::GetParameters");
ADO_Command(1)->GetParameters( ADOParameterSet[0] );
EndCheckpoint("ADOCommand::GetParameters");
```

ADO_Connect(n)->BeginTrans

Begins a new transaction.

If your provider supports nested transactions, you can call the BeginTrans method within an open transaction to start a new, nested transaction. The method returns the level of nesting: 1 indicates a top-level transaction, 2 indicates a second-level transaction, etc.

Syntax

```
ADO_Connect(n)->BeginTrans( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing the level of nesting used on this transaction.

Example


```
ADO_Connect(1)->BeginTrans( pLong );
BeginCheckpoint("ADOConnect::Execute");
ADO_Connect(1)->Execute( "DELETE FROM Temp WHERE HI =" 291667 ", pvValue, -1,
ADORecordset[7] );
```

```
EndCheckpoint("ADOConnect::Execute");
ADORecordset.Release( 7, ADOBM );
```

ADO_Connect(n)->Close

Closes a Connection object.

This method closes a Connection object and any active Recordset objects associated with the Connection. Any Command object associated with the Connection object will still exist, but will no longer be associated with a Connection object.

 **Note:** There is currently no provision to independently track the Recordsets or command associated with each different Connection object.

Syntax

```
ADO_Connect(n)->Close();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Connect( 1 )->Close( );
ADOConnect.Release( 1 );
```

ADO_Connect(n)->CommitTrans

Save changes, ends the transaction. May start a new transaction.

Only affects the most recently opened transaction. Close or rollback a transaction to resolve higher-level transactions.

After you call the BeginTrans method, the provider will no longer instantaneously commit changes you make until you call CommitTrans or RollbackTrans to end the transaction.

Syntax

```
ADO_Connect(n)->CommitTrans();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Connect(1)->BeginTrans ( pLong );
ADO_Recordset(18)->GetState ( pLong );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(FIRSTNAME)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(15)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(LASTNAME)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(11)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(ADDRESS)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(12)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
```

QALoad 5.02

```
ADO_Recordset(27)->Close();
ADORecordset.Release( 27, ADOBM );
ADO_LoadVariant( pvSource, "8", "select max(lsystemID) from CITY" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(13)->Open( pvSource, pvValue, adOpenForwardOnly, adLockReadOnly, -1 );
ADORecordset.Release( 13, ADOBM );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(19)->Update( pvValue, pvData );
ADO_Connect(1)->CommitTrans();
```

ADO_Connect(n)->Execute

Executes the query passed in the CommandText argument on the connection to the method. If a row-returning query is specified, results are stored in a new ADO Recordset object. If a row-returning query is not specified, a closed ADO Recordset object is returned.

Syntax

```
ADO_Connect->Execute( "Command", pvValue, #,ADORecord sRecordset[#] );
```

Parameters

Parameter	Description
Command	An ANSI representation of the command string.
pvValue	A pointer to the variant returning the number of records affected by this call.
#	The options used to make this call run.
ADORecordset(#)	The QALoad object containing the new recordset created by this call. The Recordset that is returned is returned inside of the identified ADORecordset(#) object.

Example

```
BeginCheckpoint("ADOConnect::Execute");
ADO_Connect->Execute( "Command", pvValue, #, ADORecordset(#) );
EndCheckpoint("ADOConnect::Execute");
ADORecordset.Release( 0, ADOBM );
```

ADO_Connect(n)->GetAttributes

GetAttributes is read/write. It's value is the sum of one or more XactAttributeEnum values. The default is zero (0).

Syntax

```
ADO_Connect(n)->GetAttributes( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 262144 );
ADO_Connect(0)->PutAttributes( 393216 );
ADO_Connect(0)->GetAttributes( pLong );
```

ADO_Connect(n)->GetCommandTimeout

Returns the value of the timeout in a pointer to a long. Use GetCommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Connect(n)->GetCommandTimeout( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->GetConnectionString

GetConnectionString method retrieves the value of ConnectionString property of this instantiation of the ADO Connect object. The ConnectionString specifies a data source by passing argument=value statements. These are separated by semi-colons.

Syntax

```
ADO_Connect(n)->GetConnectionString( sLoadStr );
```

Parameters


Parameter	Description
n	An index to the object.
sLoadStr	This CLoadString retrieves the value of the ConnectionString property for this instance of ADO Connection.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 0 );
ADO_Connect(0)->GetConnectionString( sLoadStr );
ADO_Connect(0)->PutConnectionString( "DSN=LoadTestBox;UID=SA;PWD=H" );
```

ADO_Connect(n)->GetConnectionTimeout

Use GetConnectionTimeout on a Connection object to cancel a connection attempt, if necessary, due to network or server delays. If the time interval specified in the Connection Timeout property setting runs out before a connection can be opened, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

 **Note:** Before using GetConnection Timeout, ensure that your provider supports ADO's ConnectionTimeout functionality.

Syntax

```
ADO_Connect(n)->GetConnectionTimeout( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long returning the value in seconds of the connection timeout property of this ADOConnection object.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->GetCursorLocation

Lets you choose a cursor library from those accessible to the provider. You can usually choose to use a client-side or server-side location. This setting will only affect those connections made after setting the property; it has no effect on existing connections.

Syntax

```
ADO_Connect(n)->GetCursorLocation( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Connect(0)->GetCursorLocation( pLong );
ADO_Connect(0)->PutCursorLocation( adUseServer );
ADO_Connect(0)->GetCommandTimeout( pLong );
```

ADO_Connect(n)->GetDefaultDatabase

Retrieves the value of the DefaultDatabase property from this instance of the ADO Connect object.

Syntax

```
ADO_Connect(n)->GetDefaultDatabase( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	The CLoadString which returns the DefaultDatabase information to the script.

Example

```
ADO_Connect(0)->Open("PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa; pwd="" ;database=Master;", "",
"", -1 );
ADO_Connect(0)->GetDefaultDatabase( sLoadStr );
ADO_Connect(0)->PutDefaultDatabase("pubs" );
```

ADO_Connect(n)-> GetIsolationLevel

Sets a Connection object's isolation level. Takes effect the next time the BeginTrans method is called.

Syntax

```
ADO_Connect(n)-> GetIsolationLevel( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long returning the value of the isolation level property of this instance of the ADO Connection object.

Example

```
ADO_Connect(0)->GetIsolationLevel( pLong );
ADO_Connect(0)->PutIsolationLevel( adXactSerializable );
```

ADO_Connect(n)->GetMode

Sets or returns access permissions for the current connection. The GetMode property can only be set after the Connection object is closed.

Syntax

```
ADO_Connect(n)->GetMode( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Retrieves the ConnectionModeEnum from the call and converts that to a long* to be returned to the script.

Example

QALoad 5.02

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );  
ADO_Connect(0)->GetMode( pLong );  
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );  
ADO_Connect(0)->GetProvider( sLoadStr );
```

ADO_Connect(n)->GetProvider

Returns the provider name for a connection.

You can also set this property using the contents of the Open method's ConnectionString property or argument. Note that you may get unwanted results if you specify a provider in more than one place while using the Open method. If a provider is not specified, this property will default to MSDASQL.

When the connection is closed, GetProvider is read/write. When the connection is open, GetProvider is read-only. Before the setting can take effect, you must open the Connection object or access the Connection object's Properties collection. An invalid setting results in an error.

Syntax

```
ADO_Connect(n)->GetProvider( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	This CLoadString holds the Provider information returned by this call.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );  
ADO_Connect(0)->GetMode( pLong );  
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );  
ADO_Connect(0)->GetProvider( sLoadStr );
```

ADO_Connect(n)->GetState

Determines the state of a specified object at any time. This property can have a combination of values.

Syntax

```
ADO_Connect(n)->GetState( pLong );
```

Return Value

0 for closed, 1 for open

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long holding the state information (0 or 1).

Example

```
ADO_Connect(0)->PutProvider( "MSDASQL" );
ADO_Connect(0)->GetState( pLong );
ADO_Connect(0)->GetVersion( sLoadStr );
```

ADO_Connect(n)->GetVersion

Returns the version of ADO that is being used.
The version is available as a dynamic property in the Properties collection.

Syntax

```
ADO_Connect(n)->GetVersion( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the pointer.
sLoadStr	The CLoadString string which returns version to the script.

Example

```
ADO_Connect(0)->PutProvider( "MSDASQL" );
ADO_Connect(0)->GetState( pLong );
ADO_Connect(0)->GetVersion( sLoadStr );
```

ADO_Connect(n)->Open

Establishes the connection to the data source.
When successful, it creates a live connection against which you can issue commands.

Syntax

```
ADO_Connect(n)->Open( "Connect String", "User", "Password", # );
```

Parameters

Parameter	Description
n	An index to the object.
Connect String	The ConnectionString property automatically inherits the value used for the ConnectionString argument. Therefore, you can either set the ConnectionString property of the Connection object before opening it, or use the ConnectionString argument to set or override the current connection parameters during the Open method call.
User	The UserID and Password arguments will override the values specified in ConnectionString.
Password	See the User parameter.
#	A ConnectOptionEnum value that determines whether this method should return after (synchronously) or before (asynchronously) the connection is established.

Example

```
ADO_Connect(1)->PutConnectionString( "DSN=My;UID=sa; PWD="" ;" );
BeginCheckpoint("ADOConnect::Open");
ADO_Connect(1)->Open( "", "", "", -1 );
EndCheckpoint("ADOConnect::Open");
ADO_Connect(1)->PutCommandTimeout( 200 );
```

ADO_Connect(n)->OpenSchema


Returns information about the data source. For example, tables, columns included in the tables and data types

Syntax

```
ADO_LoadVariant( pvValue, VT_TYPE, <VALUE> )
ADO_LoadVariant( pvData, VT_TYPE, <VALUE> )
ADOConnection(n)->OpenSchema( <SchemaEnum>, pvValue, pvData, ADORecordset[#] );
```

Parameters

Parameter	Description
n	An index to the object.
nSchema	SchemaEnum describing the type of schema query to run.
pvValue	An array of query constraints for each QueryType option, as listed in SchemaEnum.
pvData	This parameter is required if QueryType is set to adSchemaProviderSpecific; otherwise, it is not used.
ADORecordset[#]	The recordset composing the result set.

 **Note:** The Accompanying ADO_LoadVariant calls are made to set the proper values for the VARIANTS pvValue and pvData. They are included automatically by QALoad 's conversion process.

ADO_Connect(n)->PutAttributes

Sets the transaction attribute for this connection object. The number represented by the string in the call will be used by ADO during the call.

Syntax

```
ADO_Connect(n)->PutAttributes( XactAttributeEnum );
```

Parameters

Parameter	Description
n	An index to the object.
XactAttributeEnum	The sum of one or more XactAttributeEnum values (the default is 0). This property is read/write.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes((XactAttributeEnum) 262144 );
ADO_Connect(0)->GetAttributes( pLong );
```

ADO_Connect(n)->PutCommandTimeout

PutCommandTimeout Sends a timeout in seconds before the command will timeout with an error.

Use PutCommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Connect(n)->PutCommandTimeout( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	The number of seconds before an exception is generated by the command overrunning the timeout.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->PutConnectionString

Specifies a data source.

PutConnectionString passes detailed connection strings that include argument=value statements. Use semicolons to separate the argument=value statements.

Syntax

```
ADO_Connect(n)->PutConnectionString( CLoadString );
```

Parameters


Parameter	Description
n	An index to the object.
CLoadStr	This ANSI string sets the value of the ConnectionString property for this instance of ADO Connection.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 0 );
ADO_Connect(0)->GetConnectionString( sLoadStr );
ADO_Connect(0)->PutConnectionString( "DSN=LoadTestBox;UID=SA;PWD=H" );
```

ADO_Connect(n)->PutConnectionTimeout

Use PutConnectionTimeout on a Connection object to abandon an attempt to connect due to network or server delays. If a connection is not made in the specified time, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

 **Note:** Before using PutConnection Timeout, ensure that your provider supports ADO's ConnectionTimeout functionality.

Syntax

```
ADO_Connect(n)->PutConnectionTimeout( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	The number of seconds to attempt a connection to the provider.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->PutCursorLocation

Lets you choose a cursor library from those accessible to the provider. You can usually choose to use a client-side or server-side library. This setting will only affect those connections made after setting the property; it has no effect on existing connections.

Syntax

```
ADO_Connect(n)->PutCursorLocation( CursorLocationEnum );
```

Parameters

Parameter	Description
n	An index to the object.
CursorLocationEnum	A String representation of a CursorLocationEnum value.

Example

```
ADO_Connect(0)->GetCursorLocation( pLong );
ADO_Connect(0)->PutCursorLocation( adUseServer );
ADO_Connect(0)->GetCommandTimeout( pLong );
```

ADO_Connect(n)->PutDefaultDatabase

Sets the default database within a connection object.

Access objects in a database other than the one specified in the DefaultDatabase property by qualifying object names with the desired database name. Upon connection, the provider will write default database information to the DefaultDatabase property.

Syntax

```
ADO_Connect(n)->PutDefaultDatabase( "NAME" );
```

Parameters

Parameter	Description
n	An index to the object.
"NAME"	The name of the default database to set within the connection object.

Example

```
ADO_Connect(0)->Open( "PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa;
                      pwd="";database=Master;", "", "", -1 );
ADO_Connect(0)->GetDefaultDatabase( sLoadStr );
ADO_Connect(0)->PutDefaultDatabase("pubs" );
```

ADO_Connect(n)->PutIsolationLevel

Sets the isolation level of a Connection object. The isolation level will take effect the next time you call BeginTrans.

The isolation level that is part of this call is represented as a string. This string representation translates into a number that is used by ADO internally.

Syntax

```
ADO_Connect(n)->PutIsolationLevel( <IsolationLevelEnum> );
```

Parameters


Parameter	Description
n	An index to the object.
IsolationLevelEnum	The isolation level either in numeric format or in the adXactSerializable format. Possible values are: adXactUnspecified (value=-1), adXactChaos (value=16), adXactReadUncommitted (value=256), adXactBrowse (value=256), adXactCursorStability (value=4096), adXactReadCommitted (value=4096), adXactRepeatableRead (value=65536), adXactSerializable (value=1048576), adXactIsolated (value=1048576).

Example

```
ADO_Connect(0)->GetIsolationLevel( pLong );
ADO_Connect(0)->PutIsolationLevel( adXactSerializable );
```

ADO_Connect(n)->PutMode

Sets the access permissions being used on the current connection.

 **Note:** This property can only be set when the Connection object is closed.

Syntax

```
ADO_Connect(n)->PutMode( <ConnectionModeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
ConnectionModeEnum	Sets the ConnectionModeEnum property for this instance of the ADO Connect object.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectionModeEnum)12 );
ADO_Connect(0)->GetProvider( sLoadStr6 );
```

ADO_Connect(n)->PutProvider

Sets the provider name for a connection.

You can also set this property using the contents of the Open method's ConnectionString property or argument. Note that you may get unwanted results if you specify a provider in more than one place while using the Open method. If a provider is not specified, this property will default to MSDASQL.

When the connection is closed, PutProvider is read/write. When the connection is open, PutProvider is read-only. Before the setting can take effect, you must open the Connection object or access the Connection object's Properties collection. An invalid setting results in an error.

Syntax

```
ADO_Connect(n)->PutProvider( "Provider" );
```

Parameters

Parameter	Description
n	An index to the object.
"Provider"	The name of the provider being used to access data.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectionModeEnum)12 );
ADO_Connect(0)->PutProvider( "MSDASQL" );
```

ADO_Connect(n)->RollbackTrans

Reverses changes made in an open transaction and ends the transaction. This is linked with BeginTrans. This will only be seen in the script if a transaction fails for some reason. If it fails and you see this call, look over the script logic and see if the transaction can be committed.

Syntax

```
ADO_Connect(n)->RollbackTrans( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```

ADO_Connect(1)->BeginTrans( pLong );
ADO_Recordset(18)->GetState( pLong );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(FIRSTNAME)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(15)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(LASTNAME)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(11)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(ADDRESS)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(12)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_Recordset(27)->Close();
ADORecordset.Release( 27, ADOBM );
ADO_LoadVariant( pvSource, "8", "select max(1systemID) from CITY" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(13)->Open( pvSource, pvValue, adOpenForwardOnly, adLockReadOnly, -1 );
ADORecordset.Release( 13, ADOBM );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(19)->Update( pvValue, pvData );
ADO_Connect(1)->Rollback();

```

ADO_Field(n)->AppendChunk

A special data handling method that writes data, in chunks, to the Field object.

This is especially useful when memory is limited; you can use this method to manipulate long values in manageable chunks.

It may take numerous calls to AppendChunk to completely write the data to the appropriate field object. When writing data values using ADO, the datatype being used as the parameter with the data value is often a VARIANT datatype.

The first call to AppendChunk writes data to the field and overwrites any existing data. Subsequent calls add to the data. Note that if you append data to one field, then manipulate another field in the same record, ADO assumes you are finished with the first field. If you then attempt to append data to the first field, the existing data will be overwritten.

The AppendChunk call will be preceded immediately in the script by a call to ADO_LoadVariant. You can use the AppendChunk method in the Attributes property of a Field object if the adFldLong bit in the Attributes property is set to true.

Syntax

```
ADO_Field(n)->AppendChunk( pvValue )
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	This pointer to a variant contains a chunk of data to be written to the field

	object.
--	---------

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SUSERRESUME" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Roger Smith\n2114 Edgemere Court\nGrosse Pointe" ",
                MI 48263\n\n\nObjective:\n\n\nA job controlling the Universe"
                "which I can do from the comfort of my own home." );
ADO_Field(0)->AppendChunk( pvValue );
ADO_LoadVariant( pvValue, "8", "\n\nExperience:\nCEO for General Motors during"
                "the Reagan administration. CEO for General Electric during"
                "the Carter Administration .. \n" );
ADO_Field(0)->AppendChunk( pvValue ); /* Type: 8 - VT_BSTR Data: admin */
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724");
ADO_Recordset(18)->Update( pvValue, pvData );
```

ADO_Field(n)->GetActualSize

Retrieves the value contained within the ActualSize property of this instance of the ADO Field object. GetActualSize returns a Long value. If your provider allows this property to reserve space for BLOB data, the default is zero.

GetActualSize is read-only. If ADO fails to identify the length of the object's value, this property returns adUnknown.

Syntax

```
ADO_Field(n)->GetActualSize( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long, which contains the value of the ActualSize property of this instance of the ADO Field Object.

Example

```
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
ADO_Field(3)->GetActualSize( pLong );
```

ADO_Field(n)->GetAttributes

Retrieves the value contained within the Attributes property of this instance of the ADO Field object. The Attributes property can be the sum of one or more FieldAttributeEnum values, and it is normally read-only. It is read/write, however, if all of the following conditions are met:

- ! It is a new Field object;
- ! It has been appended to the Fields collection of a Record;
- ! The Value property for the Field has been specified; and
- ! The new Field has been added by the data provider (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->GetAttributes( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long, which contains the value of the Attributes property of this instance of the ADO Field Object.

Example

```
ADO_Field(1)->PutAttributes( 32 );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 100 );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->GetChunk

Retrieves chunks of binary or character data to an appropriate buffer.

This is a special function in that it may have to be called several times in order to properly handle the data being returned from the field object.

Used on a Field object, this method returns all or part of the object's long binary or character data. Use this method to manipulate long values in manageable chunks of data.

Since the data is being returned from this call, the ADO_LoadVariant is not called.

The returned data is assigned to a variable. If the variable size is greater than the remaining data, only the data is returned. The variable is not padded with empty spaces. An empty field returns a null value.

If more than one call to GetChunk is necessary, each subsequent call starts retrieving data from the point where the previous call left off. Note that if you retrieve data from one field, then manipulate another field, ADO assumes you are finished with the first field.

Syntax

```
ADO_Field(n)->GetChunk( #, pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
#	Length of the chunk of data being retrieved.
pvValue	Pointer to the variant holding the chunk of retrieved data.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SUSERRESUME" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetChunk( 100, pvValue );
ADO_Field(0)->GetChunk( 100, pvValue );
```

ADO_Field(n)->GetDataFormat

Retrieves an IUnknown instance describing the data format for this field.

Syntax

```
ADO_Field(n)->GetDataFormat( &pIUnknown pDataFormat );
```

Parameters

Parameter	Description
n	An index to the object.
pDataFormat	Pointer to an IUnknown interface.

Example

```
ADO_Field(1)->GetDataFormat( &pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->GetDefinedSize

Retrieves the value contained within the DefinedSize property of this instance of the ADO Field object. Used to determine the data capacity of a Field object.

Syntax

```
ADO_Field(n)->GetDefinedSize( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long with the value of the DefinedSize property of this Instance of the ADO Field object.

Example

```
ADO_Field(1)->GetDataFormat( pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 4 );
```

ADO_Field(n)->GetName

Retrieves the value contained within the Name property of this instance of the ADO Field object. Note that the actual ADO call has a BSTR as the argument; therefore, there is some data conversion occurring within this call.

GetName is normally read-only. It is read/write if all of the following conditions are met:

- ! It is a new Field object
- ! The new Field object has been appended to the Fields collection of a Record
- ! The Value property for the Field has been specified and
- ! The data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->GetName( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString. Value of the Name property for this ADO Field Object.

Example

```
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
ADO_Field(3)->GetActualSize( pLong );
```

ADO_Field(n)->GetNumericScale

Retrieves the value contained within the NumericScale property of this instance of the ADO Field object. GetNumericScale sets or returns a byte value indicating the number of decimal places to which numeric values will be resolved.

GetNumericScale is normally read-only. It is read/write, however, if all of the following conditions are met:

- ! on a new Field object
- ! the new Field object has been appended to the Fields collection of a Record
- ! the Value property for the Field has been specified and,
- ! the data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->GetNumericScale( &cUChar );
```

Parameters

Parameter	Description
n	An index to the object.
pUChar	Pointer to an unsigned character

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( &cUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->GetOriginalValue

Retrieves the value contained within the Value property of this instance of the ADO Field object before any changes were made permanent by a call to an update method.

Syntax

```
ADO_Field(n)->GetOriginalValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the VARIANT in which the original value is returned.

Example

```
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetOriginalValue( pvValue );
```

ADO_Field(n)->GetPrecision

Retrieves the value contained within the Precision property of this instance of the ADO Field object. You can use it to determine the maximum number of digits used to represent a numeric Parameter or field object.

Note that the data being returned is in the form of a pointer to an unsigned char. This is essentially a byte depicting the size of a column from 0 bytes to 256 bytes.

Syntax

```
ADO_Field(n)->GetPrecision( &cUChar );
```

Parameters

Parameter	Description
n	An index to the object.
pUChar	Pointer to an unsigned character

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( &cUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->GetProperties

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

Retrieves the complete set of properties for this particular instance of the Field object. PropertySets may change for different providers.

Syntax

```
ADO_Field(n)->GetProperties( CPropertySet* pPropertySet );
```

Parameters

Parameter	Description
n	An index to the object.

pPropertySet	Set of CProperty objects. Each CProperty object contains a single characteristic, a piece of data, which partially describes the state of a particular instance of an object.
--------------	---

Example

```
ADO_LoadVariant( pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Field(n)->GetStatus

Retrieves the value contained within the **Status** property of this instance of the ADO Field object. It takes a pointer to a long as an argument. Within this parameter, the Value of the status property of this instance of the Field object is returned.

Syntax

```
ADO_Field(n)->GetStatus( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to the value within the Status property of this instance of the ADO Field object.

Example

```
ADO_Field(2)->GetStatus( pLong );
```

ADO_Field(n)->GetType

Retrieves the value contained within the **Type** property of this instance of the ADO Field object. The Actual ADO call uses another enumerated type, **DataTypeEnum**, to handle the data types. Conversion to this type happens within the QALoad call.

GetType is read/write if all of the following conditions are met:

- ! on a new Field objects
- ! the new Field object has been appended to the Fields collection of a Record
- ! the Value property for the Field has been specified
- ! the data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->GetType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long containing the data type of the field element.

Example

```
ADO_Field(2)->PutPrecision( 0x00 );
ADO_Field(1)->GetType( pLong );
ADO_Field(1)->PutType( (DataTypeEnum)8 );
```

ADO_Field(n)->GetUnderlyingValue

Specifies the current value of a Field object in the database, after any updates to the recordset. This is the current value visible to your transaction. It may be the result of a recent update by another transaction. Note that this could be different from the OriginalValue property that was originally returned to the Recordset.

Syntax

```
ADO_Field(n)->GetUnderlyingValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the variant holding the underlying value returned from the call.

Example

```
ADO_Field(3)->PutValue( pvValue );
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
```

ADO_Field(n)->GetValue

Retrieves the value of a ADO Field object into a pointer to a Variant. The argument will handle any type of data. This can be done by using the Variant datatype. Data is retrieved into a pointer to a Variant.

Syntax

```
ADO_Field(n)->GetValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	Pointer to the value contained within the Value property of this instance of the ADO Field object.

Example

```
ADO_Field(3)->GetValue( pvValue );
LoadVariant( pvValue, "8", "T " );
ADO_Field(3)->PutValue( pvValue );
```

ADO_Field(n)->PutAttributes

The PutAttributes method call sets the value contained within the Attributes property of this instance of the ADO Field object.

This property can be the sum of one or more `FieldAttributeEnum` values. It is normally read-only; however, it can be read/write if all of the following conditions are present:

- ! It is a new `Field` object
- ! The new `Field` object has been appended to the `Fields` collection of a `Record`
- ! The `Value` property for the `Field` has been specified
- ! The new `Field` has been successfully added by the data provider (using the `Field` collection's `Update` method).

Syntax

```
ADO_Field(n)-PutAttributes (n);
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Field(1)->PutAttributes((FieldAttributeEnum)32 );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 100 );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->PutDataFormat

This updates the current value of the data format updates to the ADO Recordset.

Syntax

```
ADO_Field(n)->PutDataFormat( pIUnknown );
```

Parameters

Parameter	Description
n	An index to the object.
pIUnknown	Pointer to the <code>IUnknown</code> interface.

ADO_Field(n)->PutDefinedSize

Sets the value contained within the `DefinedSize` property of this instance of the ADO `Field` object. Note that the `DefinedSize` and `ActualSize` properties are different. For example, if a `DefinedSize` property of 25 only contained a few characters, the `ActualSize` property value would be the length of those few characters.

Syntax

```
ADO_Field(n)->PutDefinedSize( # );
```

Parameters

Parameter	Description
n	An index to the object.

#	Size of the data contained within this field.
---	---

Example

```
ADO_Field(1)->GetDataFormat( pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 4 );
```

ADO_Field(n)->PutNumericScale

Sets the value contained within the NumericScale property of this instance of the ADO Field object.

PutNumericScale also sets or returns a byte value indicating the number of decimal places to which numeric values will be resolved.

PutNumericScale is normally read-only; however, it is read/write if all of the following conditions are met:

- ! It applies to a new Field object
- ! The new Field object has been appended to the Fields collection of a Record
- ! The Value property for the Field has been specified
- ! The data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->PutNumericScale( 0x## );
```

Parameters

Parameter	Description
n	An index to the object.
0x##	Unsigned character being passed in its hexadecimal representation. Note that the range on the # is 0 - F - 0123456789ABCDEF

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( pUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->PutPrecision

Sets the value contained within the Precision property of this instance of the ADO Field object.

You can use it to determine the maximum number of digits used to represent a numeric Parameter or Field object.

Syntax

```
ADO_Field(n)->PutPrecision( 0x## );
```

Parameters

Parameter	Description
n	An index to the object.

0x##

Unsigned character being passed in its hexadecimal representation. Note that the range on the # is 0 - F - 0123456789ABCDEF

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( pUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->PutType

Sets the value contained within the Type property of this instance of the ADO Field object. There is a conversion from the long argument to a `DataTypeEnum*`, which is accomplished through a cast.

PutType is read/write for a new Field object that has been appended to a Record's Fields collection if the following conditions are met:

- ! The Value property for the Field has been specified
- ! The data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->PutType( DataTypeEnum );
```

Parameters

Parameter	Description
n	An index to the object.
DataTypeEnum	The datatype of the field element.


Example

```
ADO_Field(2)->PutPrecision( 0x00 );
ADO_Field(1)->GetType( pLong );
ADO_Field(1)->PutType( (DataTypeEnum)8 );
```

ADO_Field(n)->PutValue

Resets the value of this instance of the ADO Field object. This is the first step in updating a Recordset's value.

In order to accommodate any type of data, ADO uses the Variant datatype with PutValue.

 **Note:** This call, and all other calls that use the Variant datatypes, will have to use an accompanying Setup function call, the call to `ADO_LoadVariant`.

Syntax

```
ADO_Field(n)->PutValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the variant that returns the data contained in the ADO Field

object.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "active" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Y" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" ); // VT_ERROR;
ADO_LoadVariant( pvData, "10", "2147614724" ); // VT_ERROR;
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
ADORecordset.Release( 2, ADOBM );
```

ADO_FieldSet(0)->GetNewEnum

In order to iterate through each ADO Field in a ADO FieldSet collection, an ADOIEnumField object is returned. The GetNewEnum call on the ADO FieldSet object creates the ADO IEnumField object allowing the enumeration to take place.

Syntax

```
ADO_FieldSet(n)->GetNewEnum( ADOIEnumField[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOIEnumField	ADO IEnumField object.

Example

```
ADO_FieldSet(0)->GetNewEnum( ADOIEnumField[0] );
while( ADO_IEnumField(0)->NextField( 1, 0, ADOField[0] ) )
{
ADO_Field(0)->GetStatus( pLong );
ADOField.Release( 0 );
}
```

ADO_FieldSet(n)->Append

Append creates and appends a new Field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprises a mechanism for updating or retrieving information from a Data Provider.

Syntax

```
ADO_FieldSet(n)->Append( "<Name>", <DataTypeEnum>, <size>, <FieldAttributeEnum>, pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
Name	The name of the Field being added to the collection of fields.

DataTypeEnum	The Data type of the field being added to the collection
Size	Size of the data for the field being added to the collection
FieldAttributeEnum	An additional descriptor for the Field being added to the collection, for example, Nullable, Fixed length
pvValue	The Field's actual data in the form of a Pointer to a VARIANT.

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] );
ADO_LoadVariant( pvValue, "8", "New ColumnData" );
ADO_FieldSet(1)->Append( "testfld1" ,adBSTR, 0 , adFldUnspecified, pvValue );
```

ADO_FieldSet(n)->Append15

Append15 creates and appends a new field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprise a mechanism for updating or retrieving information from a Data Provider. The Append15 function does NOT allow the user to add the data to this ADO Field object. It creates the ADO Field object in the ADO FieldSet collection, but does not add in the data.

Syntax

```
ADO_FieldSet(n)->Append15( "<Name>", <DataTypeEnum>, <size>, <FieldAttributeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
Name	The name of the Field being added to the collection of fields.
DataTypeEnum	The Data type of the field being added to the collection
Size	Size of the data for the field being added to the collection
FieldAttributeEnum	An additional descriptor for the Field being added to the collection, for example, Nullable, Fixed length

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] );
ADO_FieldSet(1)->Append15( "testfld1", adBSTR, 0, adFldUnspecified );
ADO_LoadVariant( pvValue, "8", "testfld1" );
ADO_FieldSet(1)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "New ColumnData" );
ADO_FieldSet(1)->PutValue( pvValue );
```

ADO_FieldSet(n)->CancelUpdate

Cancels changes made to the current or new row of an ADO Recordset object, or the ADO Fieldset collection of an ADO Record object, before calling the Update method.

Syntax

```
ADO_FieldSet(n)->CancelUpdate( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

None.

ADO_FieldSet(n)->Delete

Deletes an object from the Fields collection. Takes the form of a Variant designating a Field object to delete. The Variant can be the name or ordinal position of the Field object.

Syntax

```
ADO_LoadVariant( pvValue, "Type", "Value" );
ADO_FieldSet(n)->Delete( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a Variant. The variant depicts the field to remove.

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] ); LoadVariant( pvValue, "2", "1" );
ADO_FieldSet(1)->Delete( pvValue );
ADOFieldSet.Release( 1 );
```

ADO_FieldSet(n)->GetCount

The method returns the number of ADO Field objects contained within the ADO FieldSet collection.

Syntax

```
ADO_FieldSet(n)->GetCount( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A Pointer to a long containing the number of ADO Field objects in this ADO FieldSet Collection.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
ADO_FieldSet(0)->Refresh();
```


ADO_FieldSet(n)->GetItem

This call retrieves a ADO Field object from this instance of the ADO FieldSet collection. The result of the call is that a ADO Field object is brought back to be manipulated within the script. ADO Field retrieval is a part of the variablization process.

In the example, sSSN is declared as a local variable and is a key element used in several calls to the data provider. This is variablized within the script and linked to a datapool earlier in the script.

Syntax

```
ADO_LoadVariant( pvValue, "Type", "Value" );
ADO_FieldSet(n)->GetItem( pvValue, ADOField[#] );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	This input parameter is used to pick the particular ADO Field instance from the ADO FieldSet collection.
ADOField[#]	This output parameter is used to store away the ADO Field returned by this call.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SSN" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", sSSN );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
ADORecordset.Release( 2, ADOBM );
```

ADO_FieldSet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO FieldSet collection has no visible effect.

Syntax

```
ADO_FieldSet(n)->Refresh();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
ADO_FieldSet(0)->Refresh();
```

ADO_FieldSet(n)->Resync

Synchronizes the values of a Record object's Fields collection with the data source. The Count property is not affected by this method.

Syntax

```
ADO_FieldSet(n)->Resync( <ResyncEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
ResyncEnum	This has 2 values adResyncAllValues and AdResyncUnderlying Values, either is a valid value.

Example

```
ADO_LoadVariant(pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADOField.Release( 0 );
ADO_FieldSet(0)->Update();
ADO_FieldSet(0)->Resync( adResyncAllValues );
```

ADO_FieldSet(n)->Update

Saves any changes you make to the ADO FieldSet collection of a Record object.

Syntax

```
ADO_FieldSet(n)->Update( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADOField.Release( 0 );
ADO_LoadVariar( pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "2", "15" );
ADO_Field(0)->PutValue( pvValue );
ADOField.Release( 0 );
ADO_FieldSet(0)->Update();
```

ADO_IEnum(n)->NextProperty

Enumeration through collections of properties should be done very carerfully, because in the example below, we will reset all of the properties to the same value. To reset different values, get rid of the loop and set each property individually.

Syntax

```
ADO_IEnum(n)->NextProperty( <bFetched>, <bRetrieved>, ADOProperty(0) );
```

Parameters

Parameter	Description
n	An index to the object.
Bfetched	Retrieve this property 0 FALSE 1 TRUE.
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE.
ADOProperty[#]	The ADO Property returned to be worked on.

Example

```

ADO_Connect(0)->GetProperties( ADOPROPERTYSET[0] );
ADO_PropertySet(0)->GetNewEnum( ADOIENUM[0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOPROPERTY[0] ) )
{
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADOPROPERTY.Release( 0 );
}
ADOIENUM.Release( 0 );
ADOPROPERTYSET.Release( 0 );

```

ADO_IEnumField(n)->NextField

Enumeration through collections of ADO Fields should be done very carefully, because in the example given below, the script checks the status of each of the different ADO Fields. In order to do more meaningful work, reset values of different ADO Fields then break them out of the look and use the PutValue call to place new values into the ADO Field objects.

Syntax

```
ADO_IEnumField(n)->NextField( 1, 0, ADOFIELD(0) );
```

Parameters

Parameter	Description
n	An index to the object.
Bfetched	Retrieve this property: 0 FALSE 1 TRUE
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE
ADOFIELD[n]	The instance of the ADO Field object that we are interested in.

Example

```

ADO_FieldSet(0)->GetNewEnum( ADOIENUMFIELD[0] );
while( ADO_IEnumField(0)->NextField( 1, 0, ADOFIELD[0] ) )
{
ADO_Field(0)->GetStatus( pLong );
ADOFIELD.Release( 0 );
}
ADOIENUMFIELD.Release( 0 );

```

ADO_IEnumParameter(n)->NextParameter

Enumeration through collections of ADO Parameters should be done very carefully, because in the example given below, the script checks different values of each of the different ADO Parameters. In order to do some more meaningful work, resetting values of different ADO Parameters then break them out of the look and use the PutValue call to place new values into the ADO Parameter objects.

Syntax

```
ADO_IEnumParameter(n)->NextParameter( 1, 0, ADOParameter[0] );
```

Parameters

Parameter	Description
n	An index to the object.
Bfetched	Retrieve this property 0 FALSE 1 TRUE
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE
ADOParameter[n]	The ADO Parameter being returned to be worked on

Example

```
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
ADO_ParameterSet(0)->GetNewEnum( ADOIEnumParameter[0] );
while( ADO_IEnumParameter(0)->NextParameter( 1, 0,
ADOParameter[0] ) )
{
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADOParameter.Release( 0 );
}
ADOIEnumParameter.Release( 0 );
```

ADO_LoadVariant(n)

Loads the value sValue, of type sType, into the Variant structure.

Syntax

```
ADO_LoadVariant ( <VariantName>, "<Type>", "<Value>" );
```

Parameters

Parameter	Description
n	An index to the object.
VariantName	The variable that is being set up by this method.
Type	The VT_TYPE of the data. The VT_TYPE is the key to the information contained within any Variant data structure. It reveals which of the data elements within the variant structure contains data.
sValue	The actual data that will be placed into whatever VT_TYPE is called for.
ADOCommand[#]	A pointer to a specific instance of ADO Command.

Example

```
ADO_Connect(0)->GetProperties( ADOPropertySet[0] );
ADO_PropertySet(0)->GetNewEnum( ADOIEnum[0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOProperty[0] ) )
{
  ADO_Property(0)
```

ADO_Parameter(n)->AppendChunk

A special data handling method that writes data, in chunks, to the Parameter object.

This is especially useful when memory is limited; you can use this method to manipulate long values in manageable chunks.

It may take numerous calls to AppendChunk to completely write the data to the appropriate object. When writing data values using ADO, the datatype being used as the parameter with the data value is often a VARIANT datatype.

The first call to AppendChunk writes data to the parameter and overwrites any existing data. Subsequent calls add to the data. Note that if you append data to one parameter, then manipulate another parameter in the same record, ADO assumes you are finished with the first parameter. If you then attempt to append data to the first parameter, the existing data will be overwritten.

The AppendChunk call will be preceded immediately in the script by a call to ADO_LoadVariant.

You can use the AppendChunk method in the Attributes property of a parameter object if the adFldLong bit in the Attributes property is set to true.

Syntax

```
ADO_LoadVariant( pvValue, "TypeString", "ValueString" );
ADO_Parameter(n)->AppendChunk( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The Variant containing the chunk of data to send to the ADO_Parameter object instance.

Example

```
ADO_Parameter(0)->PutType( adBSTR );
ADO_LoadVariant( pvValue, "8", "a big chunk of data" );
BeginCheckpoint("ADODataBinder::AppendChunk");
ADO_Parameter(0)->AppendChunk( pvValue );
EndCheckpoint("ADODataBinder::AppendChunk");
ADO_LoadVariant( pvValue, "8", "some more data" );
BeginCheckpoint("ADODataBinder::AppendChunk");
ADO_Parameter(0)->AppendChunk( pvValue );
EndCheckpoint("ADODataBinder::AppendChunk");
```

ADO_Parameter(n)->GetAttributes

Retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->GetAttributes( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing the attribute or Direction value of the Parameter object.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
```

ADO_Parameter(n)->GetDirection

Retrieves the value contained within the Direction property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->GetDirection( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing the attribute or Direction value of the Parameter object.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
```

ADO_Parameter(n)->GetName

Retrieves the value contained within the Name property of this instance of the ADO Parameter object. GetName is read/write for Parameter objects that haven't been appended to the Parameters collection. It is read-only for appended Parameter objects, and all other objects. Note that, within a collection, names do not have to be unique.

Syntax

```
ADO_Parameter(n)->GetName( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString value being returned with the name of the Parameter.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
ADO_Parameter(0)->GetPrecision( pUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
ADO_Parameter(0)->GetValue( pvValue );
```

ADO_Parameter(n)->GetNumericScale

Retrieves the value contained within the NumericScale property of this instance of the ADO Parameter object. Returns a Byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

Syntax

```
ADO_Parameter(n)->GetNumericScale( &cUChar );
```

Parameters

Parameter	Description
n	An index to the object.
&cUchar	The address of an unsigned character.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(0)->GetPrecision( &cUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
```

ADO_Parameter(n)->GetPrecision

Retrieves the value contained within the Precision property of this instance of the ADO Parameter object. Returns a Byte value showing the maximum number of digits used to represent values for a numeric Parameter object. The Precision property is read/write.

Syntax

```
ADO_Parameter(n)->GetPrecision( &cUChar );
```

Parameters

Parameter	Description
n	An index to the object.
&cUchar	The address of an unsigned character.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
```

```
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(0)->GetPrecision( &cUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
```

ADO_Parameter(n)->GetSize

Retrieves the value contained within the **Size** property of this instance of the ADO Field object.

GetSize indicates the maximum size of a Parameter object, in bytes or characters. You can use it to determine the maximum size for values of Parameter object's **Value** property.

If the data type specified for a Parameter object is of variable length, set the object's **Size** property before appending it to the Parameters collection. If you do not, an error occurs.

Syntax

```
ADO_Parameter(n)->GetSize( pLong );
```

Parameter

Parameter	Description
n	An index to the object.
pLong	A pointer to a long containing the maximum size of the parameters data.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
ADO_Parameter(0)->GetPrecision( pUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
ADO_Parameter(0)->GetValue( pvValue );
```

ADO_Parameter(n)->GetValue

Use the **Value** property to return data from ADO Parameter objects and to return parameter values with ADO Parameter objects.

Syntax

```
ADO_Parameter(n)->GetValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	Pointer to a variant used to retrieve data from the Parameter object.

Example

```
ADO_Parameter(2)->PutSize( 8 );
ADO_Parameter(2)->PutType( adDouble );
```



```
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
```

ADO_Parameter(n)->PutAttributes

This method call retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

PutAttributes is read/write. It's value can be the sum of one or more ParameterAttributesEnum values. The default is adParamSigned.

The following are ParameterAttributesEnum values:

- ! adParamSigned
- ! adParamNullable
- ! adParamLong

Syntax

```
ADO_Parameter(n)->PutAttributes( <ParameterAttributesEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
ParameterAttributesEnum	Special enumerated data type used.

Example

```
ADO_Command(0)->Execute( pvValue, pvSource, -1,
ADORecordset[0] );
ADO_Parameter(0)->PutAttributes( adParamLong );
ADO_Parameter(0)->PutType( adBSTR );
```

ADO_Parameter(n)->PutDirection

Indicates Parameter type: input, output, input and output, or the return value from a stored procedure. This method call sets the value contained within the Direction property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->PutDirection( ParameterDirectionEnum );
```

Parameters

Parameter	Description
n	An index to the object.
ParameterDirectionEnum	<p>Possible values are:</p> <ul style="list-style-type: none"> adParamInput (value=1): The default indicating the parameter represents an input parameter. adParamInputOutput (value=3): The parameter represents both an input and output parameter. adParamOutput (value=2): The parameter represents an output parameter.

	adParamReturnValue (value=4): The parameter represents a return value. adParamUnknown (value=0): The parameter direction is unknown.
--	---

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(1)->PutDirection( adParamOutput );
```

ADO_Parameter(n)->PutName

Sets the value contained within the Name property of this instance of the ADO Parameter object. PutName is read/write for Parameter objects that are not appended to the Parameters collection. It is read-only for appended Parameter objects and all other objects. Note that, within a collection, names do not have to be unique.

Syntax

```
ADO_Parameter(n)->PutName( "NameString" );
```

Parameters

Parameter	Description
n	An index to the object.
"NameString"	A string containing the name of the parameter.

Example

```
ADO_Parameter(2)->PutAttributes( 128 );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
```

ADO_Parameter(n)->PutNumericScale

Sets the value contained within the NumericScale property of this instance of the ADO Parameter object. Sends a byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

Syntax

```
ADO_Parameter(n)->PutNumericScale( 0x## );
```

Parameters

Parameter	Description
n	An index to the object.
0x##	This byte value sets the numeric scale.

Example

```
ADO_Parameter(2)->PutAttributes( adParamLong );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
```

ADO_Parameter(n)->PutPrecision

Sets the value contained within the Precision property of this instance of the ADO Parameter object. Sends a byte value showing the maximum number of digits used to represent values for a numeric ADO Parameter object. The Precision property is read/ write.

Syntax

```
ADO_Parameter(n)->PutPrecision( 0x## );
```

Parameters

Parameter	Description
n	An index to the object.
0x##	This byte value sets the precision.

Example

```
ADO_Parameter(2)->PutAttributes( adParamLong );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
```

ADO_Parameter(n)->PutSize

Retrieves the value contained within the Size property of this instance of the ADO Parameter object. Specifies the maximum size of a Parameter object, in bytes or characters. You can use it to determine the maximum size for values of a Parameter object's Value property. If the data type specified for a Parameter object is of variable length, set the object's Size property before appending it to the Parameters collection. If you do not, an error occurs.

Syntax

```
ADO_Parameter(n)->PutSize( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	A long integer representation of the size of the parameter.

Example

```
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
```

```
ADO_Parameter(1)->PutSize( 4 );
ADO_Parameter(1)->PutType(adInteger );
```

ADO_Parameter(n)->Put Type

Sets the value contained within the Type property of this instance of the ADO Parameter object.

PutType is read/write when each of the following conditions are present:

- ! On a new Parameter object
- ! The new Parameter object has been appended to a Record's Fields collection
- ! The Parameter's Value property has been specified
- ! The data provider has added the new Parameter (using the Parameters collection's Update method).

Syntax

```
ADO_Parameter(n)->PutType( <DataTypeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
DataTypeEnum	These are the various different types of data that are valid parameter types.

Example

```
ADO_Parameter(2)->PutAttributes( 128 );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
```

ADO_Parameter(n)->Put Value

Sets the value contained within the Value property of this instance of the ADO Parameter object.

PutValue can be used to set or return data from ADO Parameter objects, parameter values with ADO Parameter objects, or property settings with Property objects.

Syntax

```
ADO_Parameter(n)->PutValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The value being loaded into this parameter.

Example

```
ADO_Parameter(2)->PutSize( 8 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
```

ADO_ParameterSet(n)->Append

Appends a ADO Parameter object to the collection of ADO Parameters.

As in the example below, the ADO Parameters are created and given a value using CreateParameter calls, then the ADO Parameters are Appended to the ADO ParameterSet container.

Syntax

```
ADO_ParameterSet(n)->Append( ADOParameter[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOParameter[#]	ADO Parameter object being added to this collection.

Example

```
ADO_LoadVariant( pvValue, "8", "ParameterData0" );
ADO_Command(0)->CreateParameter( "Param3", adInteger, adParamInput,
0, pvValue, ADOParameter[0] );
ADO_LoadVariant( pvValue, "8", "ParameterData1" );
ADO_Command(0)->CreateParameter( "Param4", adInteger, adParamInput,
0, pvValue, ADOParameter[1] );
ADO_ParameterSet(0)->Append( ADOParameter[0] );
ADO_ParameterSet(0)->Append( ADOParameter[1] );
```

ADO_ParameterSet(n)->Delete

Deletes an ADO Parameter object from the ADO ParameterSet collection.

Takes the form of a Variant designating an ADO Parameter object to delete. The Variant can be the name or ordinal position of the ADO Parameter object.

Syntax

```
ADO_ParameterSet(n)->Delete( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a variant containing information describing the ADO Parameter to be deleted from the ADO ParameterSet collection.

Example

```
ADO_LoadVariant( pvValue, "2", "3" );
BeginCheckpoint( "ADOParameterSet::Delete" );
ADO_ParameterSet(0)->Delete( pvValue );
EndCheckpoint( "ADOParameterSet::Delete" );
```

ADO_ParameterSet(n)->GetCount

The method returns the number of ADO Parameter objects contained within the ADO ParameterSet collection.

Syntax

```
ADO_ParameterSet(n)->GetCount( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long containing the number of ADO Parameter objects in this ADO ParameterSet Collection.

Example

```
ADO_ParameterSet(0)->GetCount( pLong );
ADO_ParameterSet(0)->Refresh();
```

ADO_ParameterSet(n)->GetItem

Locates a specific ADO Parameter in the ADO ParameterSet collection.

An ADO ParameterSet collection is an array of ADO Parameter objects. GetItem indexes through the array to locate a specific object.

Syntax

```
ADO_LoadVariant( pvValue, "Type", "Value" );
ADO_ParameterSet(n)->GetItem( pvValue, ADOParameter[#] );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The variant contains information about the parameter to retrieve from the collection. ADOParameter[#] The retrieved parameter.

Example

```
BeginCheckpoint( "ADOParameterSet::GetItem" );
ADO_ParameterSet(0)->GetItem( pvValue, ADOParameter[0] );
EndCheckpoint( "ADOParameterSet::GetItem" );
```

ADO_ParameterSet(n)->GetNewEnum

In order to iterate through all of the ADO Parameters in an ADO ParameterSet collection, an ADOEnumParameter object is returned. The GetNewEnum call on the ADO ParameterSet object creates the ADO IEnumParameter object allowing the enumeration to take place.

Syntax

```
ADO_ParameterSet(n)->GetNewEnum( ADOEnumParameter[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOEnumParameter	ADO IEnumParameter object.

Example

```
ADO_ParameterSet(0)->GetNewEnum( ADOEnumParameter[0] );
while( ADO_IEnumParameter(0)->NextParameter( 1, 0, ADOParameter[0] ));
{
ADO_Parameter(0)->GetStatus( pLong );
ADOParameter.Release( 0 );
}
```

ADO_ParameterSet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO ParameterSet collection has no visible effect.

Syntax

```
ADO_ParameterSet(n)->Refresh();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_ParameterSet(0)->GetCount( pLong );
ADO_ParameterSet(0)->Refresh();
```

ADO_Property(n)->GetAttributes

Describes column characteristics by setting or returning a Long value.

The value indicates characteristics of the table represented by the Column object. It can be a combination of ColumnAttributesEnum constants. The default value is zero (0).

Syntax

```
ADO_Property(n)->GetAttributes( pLong );
```

Parameters

Parameter	Description
-----------	-------------

n	An index to the object.
pLong	A pointer to a long integer containing the value of the Attributes property.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );

/* Type:8 - VT_BSTR Data: Master */

ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetName

Retrieves the value of the Name attribute of this instance of the Property object.

Syntax

```
ADO_Property(n)->GetName( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	Value of the Name property for this instance of the ADO Property.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetType

Indicates a property's type as conveyed as a DataTypeEnum.

Syntax

```
ADO_Property(n)->GetType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.

pLong	A pointer to a long containing the DataTypeEnum value for the property.
-------	---

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetValue

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

 **Note:** You can use the Value property to set and return long binary data.

Syntax

```
ADO_Property(n)->GetValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A Pointer to a variant in which the value of this property will be returned.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->PutAttributes

Describes column characteristics by returning a long value, which indicates characteristics of the table represented by the Column object. The value can be a combination of ColumnAttributesEnum constants. The default value is zero (0), which is neither adColFixed nor adColNullable.

Syntax

```
ADO_Property(n)->PutAttributes( # );
```

Parameters

Parameter	Description
n	An index to the object.

#	A long integer value that should reflect one of the following: adPropNotSupported (value=0), adPropRequired (value=1), adPropOptional (value=2), adPropRead (value=512), adPropWrite (value=1024), or a combination of them.
---	---

ADO_Property(n)->PutValue

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

 **Note:** You can use the Value property to set and return long binary data.

Syntax

```
ADO_LoadVariant( pvValue, "Type", "Value" );
ADO_Property(n)->PutValue( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the VARIANT retrieves the Value of the property that is in the get, and the value is set in the Put call.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_PropertySet(n)->GetCount

The method returns the number of ADO Property objects contained within the ADO PropertySet collection.

Syntax

```
ADO_PropertySet(n)->GetCount( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long containing the number of ADO Property objects in this ADO PropertySet Collection.

Example

```
ADO_PropertySet(0)->GetCount( pLong );
ADO_PropertySet(0)->Refresh();
```

ADO_PropertySet(n)->GetItem

Retrieves a specific ADO Property in the ADO PropertySet collection.

An ADO PropertySet collection is an array of ADO Property objects. GetItem indexes through the array to locate a specific object.

Syntax

```
ADO_PropertySet(n)->GetItem( pvValue, ADOProperty );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a variant describing the property to be retrieved.
ADOProperty	An instance of an ADO Property object.

Example

```
ADO_LoadVariant( pvValue, "2", "3" );
ADO_PropertySet(0)->GetItem( pvValue, ADOProperty[0] );
```

ADO_PropertySet(n)->GetNewEnum

In order to iterate through all of the ADO Propertys in an ADO PropertySet collection, an ADO IEnum object is returned. The GetNewEnum call on the ADO PropertySet object creates the ADO IEnum object allowing the enumeration to take place.

Syntax

```
ADO_PropertySet(n)->GetNewEnum( ADOIEnum [#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOIEnum	ADO IEnum object.

Example

```
ADO_PropertySet(0)->GetNewEnum( ADOIEnum [0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOProperty[0] ) )
{
ADO_Property(0)->GetStatus( pLong );
ADOProperty.Release( 0 );
}
```

ADO_PropertySet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO PropertySet collection has no visible effect.

Syntax

```
ADO_PropertySet(n)->Refresh();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_PropertySet(0)->GetCount( pLong );
ADO_PropertySet(0)->Refresh();
```

ADO_Record(n)->Cancel

Cancels execution of a pending, asynchronous method call.

Syntax

```
ADO_Record(n)->Cancel();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Record(1)->CopyRecord( "Home", "Away", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
```

ADO_Record(n)->Close

Use to close a Recordset, Record, or Stream object. Any associated data or exclusive access you may have had to the data through this particular object will be released. You can reopen the object later using the Open method.

Syntax

```
ADO_Record(n)->Close();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Record(1)->CopyRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->CopyRecord

Copies a file or directory (including its contents) to another location.

Tip: Ensure that the values of Source and Destination are not identical or you will receive a run-time error. One of the server, path, or resource names must differ.

All subdirectories are copied recursively, unless `adCopyNonRecursive` is specified. In a recursive operation, Destination must not be a subdirectory of Source; otherwise, the operation will not be able to finish. Insert your product name (QALoad , for example)'s implementation of the CopyRecord method makes the call through to the CopyRecord method within the ADO Record object.

Note that the CopyRecordOptionsEnum is often in the form of a number. This occurs when the CopyRecordOptionsEnum is formed from a combination of values.

Syntax

```
ADO_Record(n)->CopyRecord( ( "Source", "Target", "User", "Password",
                             CopyRecordOptionsEnum, Async ) );
```

Parameters

Parameter	Description
n	An index to the object.
Source	String value containing a URL that specifies the entity that is to be copied.
Target	String value containing a URL that specifies the location to which the Source will be copied.
User	This is the user name used to determine whether a particular user has permission to use this information.
Password	A String value. The password for the particular user to verify that the user has permission to perform the operation.
CopyRecordOptionsEnum	Copy options.
Async	If this is an asynchronous operation.

Example

```
ADO_Record(1)->CopyRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->DeleteRecord

Deletes a file or directory, and all its subdirectories.

After this method is finished, any operations on the file or directory represented by this Record could fail. Close the Record after calling this method.

Insert your product name (QALoad , for example)'s implementation of the DeleteRecord method makes the call through to the DeleteRecord method within the ADO Record object.

Syntax

```
ADO_Record(n)->DeleteRecord( "Source", # );
```

Parameters

Parameter	Description
n	An index to the object.
Source	A string value that contains a URL identifying the entity (for example, the file or directory) to be deleted.
#	Is this an asynchronous call (-1 TRUE, 0 FALSE)

Example

```
ADO_Record(1)->DeleteRecord( "\\QAServer\\Temp\\GeoffR", 0 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->GetActiveConnection

Use the ActiveConnection property to determine the ADO Connect object over which the specified ADO Record object will execute.

Syntax

```
ADO_Record(n)->GetActiveConnection( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A Pointer to a Variant containing the ADO Connect object.

Example

```
LoadVariant( pvValue, ADOConnect[1] );
ADO_Record(1)->GetActiveConnection( pvValue );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->GetChildren

Returns an ADO Recordset, in the form of a Pointer to an ADO Recordset object, whose rows represent the files and subdirectories in the directory represented by this Record.

Syntax

```
ADO_Record(n)->GetChildren( ADORecordset[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADORecordset[#]	This is the information retrieved from this call. The GetChildren call retrieves the data into a ADO Recordset pointer.

Example

```
ADO_Record(1)->GetChildren( ADORecordset[1] );
ADO_Record(1)->Close();
```

ADO_Record(n)->GetFields

Contains all the Field objects of an ADO Recordset or ADO Record object.

Insert your product name (QALoad , for example)'s implementation of the GetFields method takes care of making the call through to the GetFields method within the ADO Record object. In this call, the ADO Fields object that is returned is wrapped within the ADO FieldSet object.

Syntax

```
ADO_Record(n)->GetFields( ADOFieldSet[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOFieldSet[#]	Set of fields that compose the record.

Example

```
ADO_Record(1)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "dsn_name" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetValue( pvValue ); /* Type: 8 - VT_BSTR Data: FOCFG */
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_Record(1)->MoveNext();
ADO_Record(1)->GetEOF( pVTBOOL );
ADO_Record(1)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "dsn_name" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetValue( pvValue ); /* Type: 8 - VT_BSTR Data: FOCRFP */
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
```

ADO_Record(n)->GetMode

Sets or returns the access permissions being used on the current connection by the provider. Note that you can only set this property when the Connection object is closed.

Syntax

```
ADO_Record(n)->GetMode( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Retrieves the ConnectionModeEnum from the call and converts that to a long* to be returned to the script.

Example

```
ADO_Record(1)->GetMode( pLong );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->GetParentURL

Sets the current value of the source property for this instance of the actual ADO Command object. This property depends on which source is used to open the Record object.

Syntax

```
ADO_Record(n)->GetParentURL( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	CLoadString holding the parent URL retrieved by the call.

Example

```
ADO_Record(1)->PutSource( "\\\\QAserver\\MyDirectory" );
ADO_Record(1)->GetParentURL( sLoadStr );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->GetRecordType

This method is used to check the contents of the ADO RecordType property for this instance of ADO Record object, returning the RecordTypeEnum in a pointer to a long.

Syntax

```
ADO_Record(n)->GetRecordType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing a RecordTypeEnum value.
RecordTypeEnum	Contains the following values: adSimpleRecord (value=0), adCollectionRecord (value=1), adStructDoc (value=2).

Example

```
ADO_Record(1)->PutSource( "\\\\QAserver\\MyDirectory" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```


ADO_Record(n)->GetSource

Indicates the entity represented by the ADO Record object.

The GetSource method retrieves the current value of the source property of the ADO Record object.

Syntax

```
ADO_Record(n)->GetSource( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	Variant holding the value of the source of the Record object.

Example

```
ADO_Record(1)->GetSource( pvValue );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->GetState

You can use the State property to determine the state of a given ADO Record object at any time.

Syntax

```
ADO_Record(n)->GetState( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long integer holding the state of the Record object.

Example

```
ADO_Record(1)->GetState( pLong );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->MoveRecord

Moves a file, or a directory and its contents, to another location.

A run-time error will occur if the values of Source and Destination are the same. At least one of the server, path, and resource names must differ.

All hypertext links will be updated unless otherwise specified by Options. If an existing file or directory is identified, this method will fail unless you specify adMoveOverWrite.

Note that the MoveRecordOptionsEnum is often in the form of a number. This occurs when the MoveRecordOptionsEnum is formed from a combination of values.

Syntax

```
ADO_Record(n)->MoveRecord( "Source", "Target", "User", "Password", MoveRecordOptionsEnum, Async );
```

Parameters

Parameter	Description
n	An index to the object.
Source	String value containing a URL that specifies the entity that is to be copied.
Target	String value containing a URL that specifies the location to which the Source will be copied.
User	This is the user name used to determine whether a particular user has permission to use this information.
Password	A String value. The password for the particular user to verify that the user has permission to perform the operation.
CopyRecordOptionsEnum	Copy options.
Async	If this is an asynchronous operation.


Example

```
ADO_Record(1)->MoveRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->Open

Makes the call through to the Open method within the ADO Record object to open an existing ADO Record object, or create a new file or directory.

If the entity represented by the Record object can't be accessed with a URL, the values of the ParentURL property and the field accessed with the adRecordURL constant will be null.

 **Note:** The two SetupVariantValue calls must be present. They also present opportunities for variablization of the scripts.

Syntax

```
ADO_LoadVariant( pvSource, "Type", "Data" );
ADO_LoadVariant( pvValue, "Type", "Data" );
ADO_Record(n)->Open(pvSource, pvValue, ConnectModeEnum, RecordCreateOptionsEnum, #, "sUser", "sPassword" );
```

Parameters

Parameter	Description
n	An index to the object.
Source	A pointer to a variant that may represent the URL of the entity to be represented by this ADO Record object, an ADO Command, an open ADO

	Recordset or another ADO Record object, a string containing a SQL SELECT statement or a table name.
ActiveConnection	A pointer to a variant that represents the connect string or open ADO Connect object.
Mode	A ConnectModeEnum value, whose default value is adModeUnknown, that specifies the access mode for the resultant Record object.
CreateOptions	A RecordCreateOptionsEnum value, whose default value is adFailIfNotExists, that specifies whether an existing file or directory should be opened, or a new file or directory should be created.
Options	A RecordOpenOptionsEnum value, whose default value is adOpenRecordUnspecified, that specifies options for opening the ADO Record.
UserName	A String value that contains the user ID that, if needed, authorizes access to Source.
Password	A String value that contains the password that, if needed, verifies UserName.

Example

```
ADO_LoadVariant( pvSource, "8", "\\QAServer\ MyDirectory" );
ADO_LoadVariant( pvValue, "8", "\\QAServer\ BossDirectory" );
ADO_Record(1)->Open( pvSource, pvValue, adModeReadWrite, adCreateCollection, adOpenOutput,
"sa", "sa" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutActiveConnection

PutActiveConnection is read/write when the ADO Record object is closed. It may contain a connection string or reference to an open ADO Connect object. When the ADO Record object is open and contains a reference to an open ADO Connect object, PutActiveConnection is read-only.

Syntax

```
ADO_Record(n)->PutActiveConnection( "Connection" );
```

Parameters

Parameter	Description
n	An index to the object.
Connection	A Connection string.

Example

```
ADO_Record(1)->PutActiveConnection( "DSN=QAServer; UID=sa; PWD=sa" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutMode

Sets the access permissions being used on the current connection by the provider. You can only set this property when the ADO Connect object is closed.

Syntax

```
ADO_Record(n)->PutMode( <ConnectModeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Record(1)->PutMode( adModeShareDenyNone );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutRefActiveConnection

Specifies the ADO Connect object to be affected by the specified ADO Record object. The Argument being passed to this call is an ADO Connect. This is resolved through the ADOConnect[#] operator call.

In the example below the ADO Record is associating itself with ADO Connect object index 2.

Syntax

```
ADO_Record(n)->PutRefActiveConnection( ADOConnect[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOConnection	An existing instance of an ADO Connect object.

Example

```
ADO_Record(1)->PutRefActiveConnection( ADOConnect[2] );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutSource

Sets the current value of the source property for this instance of the actual ADO Command object. The Source property must refer to an object existing within the scope of that ADO Connect.

The Source property returns the Source argument of the ADO Record object Open method. It can contain an absolute or relative URL string. An absolute URL can be used without setting the ActiveConnection property to directly open the ADO Record object. An implicit ADO Connect object is created in this case.

Syntax

```
ADO_Record(n)->PutSource( "Source" );
```

Parameters

Parameter	Description
n	An index to the object.
Source	A string representation of an absolute or relative URL.

Example

```
ADO_Record(1)->PutSource( "\\QAServer\\Development Home.htm" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Recordset(n)->_xClone

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it makes a clone of the calling ADO Recordset. This is given the arguments and the method name.

 **Note:** Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xClone( ADORecordset[#], ADOBM );
```

Parameters

Parameter	Description
n	An index to the object.
ADORecordset[#]	This is the new instance of the ADO Recordset cloned from the calling of this method.
ADOBM	The global container of ADO Bookmarks.

Example

```
ADO_Recordset(1)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
ADO_Recordset(1)->GetEOF( pVTBOOL );
ADO_Recordset(1)->_xClone( ADORecordset[2], ADOBM );
```

ADO_Recordset(n)->_xResync

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it re-synchronizes the ADO Recordset with the underlying data provider. This is given the arguments and the method name.

 **Note:** Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xResync( <AffectEnum> );
```

Parameters

Parameter	Description
n	An index to the object.

AffectEnum	adAffectCurrent, adAffectGroup, adAffectAll, AdAffectAllChapters: are all of the different values that could be present as an argument.
------------	---

Example

```
ADO_Connect(1)->Execute( "DELETE FROM MyTemp", pvValue, -1, ADORecordset[4] );
ADO_Recordset(4)->_xResync( adAffectAll );
```

ADO_Recordset(n)->_xSave

This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it saves ADO Recordset data to the location given in the first argument. This is given the arguments and the method name.

 **Note:** Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xSave( "<String>", <PersistEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
String	The file to which the ADO Recordset information is being saved.
PersistEnum	adPersistXML, adPersistADTG are the two PersistEnum values.

ADO_Recordset(n)->AddNew

Creates a new record for an updatable ADO Recordset object. After AddNew is called, the new record becomes current and remains so after you call the Update method. If the ADO Recordset object doesn't support bookmarks, you may not be able to access the new record after moving to another record. You may need to call the Requery method to make the new record accessible.

In the example below, an empty row is being added to the end of the just opened ADO Recordset.

Syntax

```
ADO_Recordset(n)->AddNew( pvSource, pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
PvSource	A pointer to a Variant containing an Array of fields that compose the ADO Recordset.
pvValue	A pointer to an array of values corresponding to the array of fields.

Example

```
ADO_LoadVariant( pvSource, "8", "select sPhone, sExtension,
sDescription" ", iRecordID, sStudentID from PHONE
where "sStudentID='S123456'" );
```

```

LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic,
                        adLockBatchOptimistic, -1 );
ADO_Recordset(27)->GetEOF( pVTBOOL );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Recordset(27)->AddNew( pvSource, pvValue );

```

ADO_Recordset(n)->Cancel

Cancels execution of a pending, asynchronous method call.

Syntax

```
ADO_Recordset(n)->Cancel();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```

ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_LoadVariant( pvSource, "8", "SELECT * FROM test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Cancel();

```

ADO_Recordset(n)->CancelBatch

Cancels any pending updates in an ADO Recordset that is in batch update mode.

If the ADO Recordset is in immediate update mode, and you call CancelBatch without adAffectCurrent, an error results.

Syntax

```
ADO_Recordset(n)->CancelBatch( <AffectEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
AffectEnum	adAffectCurrent, adAffectGroup, adAffectAll, AdAffectAllChapters: are all of the different values that could be present as an argument.

Example

```

ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic, adLockBatchOptimistic, -1 );
ADO_Recordset(27)->CancelBatch(adAffectCurrent );

```

ADO_Recordset(n)->CancelUpdate

Cancels any changes made to the current row or discards a new row of an ADO Recordset object before calling the Update method.

You can only cancel changes to a current or new row after calling the Update method under the following conditions:

- ! The changes are part of a transaction that you can roll back with the RollbackTrans method, or
- ! The changes are part of a batch update.

Syntax

```
ADO_Recordset(n)->CancelUpdate( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic, adLockBatchOptimistic, -1 );
ADO_Recordset(27)->CancelUpdate();
```

ADO_Recordset(n)->Clone

Duplicates an ADO Recordset object. Can specify that the clone be read-only.

Use to create duplicate ADO Recordset objects, especially if you want to maintain more than one current record in a given set of records. Using this method is more efficient than creating and opening a new ADO Recordset object with the same definition as the original.

Syntax

```
ADO_Recordset(n)->Clone( <LockTypeEnum>, ADORecordset[#], ADOBM );
```

Parameters

Parameter	Description
n	An index to the object.
LockTypeEnum	This is the type of locking that should occur to the recordset while the cloning operation is taking place.
ADORecordset[#]	This is the new instance of the ADO Recordset cloned from the calling ADO_Recordset(n).
ADOBM	The globally available container of LoadBookmarks.

Example

```
ADO_Recordset(1)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
ADO_Recordset(1)->GetEOF( pVTBOOL );
ADO_Recordset(1)->Clone(adLockOptimistic, ADORecordset[2], ADOBM );
```

ADO_Recordset(n)->Close

Closes an open object and any dependent objects.

When used to close an ADO Recordset, releases the associated data and any exclusive access you may have had to the data through this object.

ActiveX Data Objects (ADO) comprises a series of objects, which have states. In the ADO Recordset and ADO Connect objects it is important to close the object before releasing the object.

Syntax

```
ADO_Recordset(n)->Close();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * FROM Test_Table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Close();
ADORecordset.Release( 0, ADOBM );
```

ADO_Recordset(n)->CompareBookmarks

Compares two bookmarks. Returns an indication of their relative values.

Compared bookmarks must apply to the same ADO Recordset object, or an ADO Recordset object and its clone. Bookmarks from different ADO Recordset objects can't be compared reliably, even when created from the same source or command. An ADO Recordset object's underlying provider must support comparisons, or you won't be able to compare bookmarks.

Syntax

```
ADO_Recordset(n)->CompareBookmarks( ADOBM[#], ADOBM[#], pLong );
```

Parameters

Parameter	Description
n	An index to the object.
ADOBM[#]	A pointer to a CLoadBookmark.
ADOBM[#]	A pointer to a CLoadBookmark.
pLong	A pointer to a long, containing the return value.

Example

```
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
BeginCheckpoint("ADORecordset::CompareBookmarks");
ADO_Recordset(0)->CompareBookmarks( ADOBM[0], ADOBM[0], pLong );
EndCheckpoint("ADORecordset::CompareBookmarks");
```

ADO_Recordset(n)->Delete

Use to delete the current record or a group of records.

This method marks the current record or a group of records in an ADO Recordset object for deletion. If the object does not allow record deletion, an error occurs. In immediate update mode, deletions occur immediately.

Syntax

```
ADO_Recordset(n)->Delete( <AffectEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
<AffectEnum>	The Parameter will vary in representation between the string representation and a pure numeric representation.

Example

```
ADO_Recordset(0)->AddNew( pvSource, pvValue );
EndCheckpoint("ADORecordset::AddNew");
BeginCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Find("test_number = 99", 0, adSearchForward, ADOBM[2] );
EndCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Delete( adAffectCurrent );
```

ADO_Recordset(n)->Find

Locates a row in an ADO Recordset that matches specified criteria.

You may specify the search direction, starting row, and offset from the starting row.

When the criteria is met, the found record becomes the current row position. If not, the current row position is set to the end or start of the ADO Recordset.

Syntax

```
ADO_Recordset(n)->Find ("criteria," <SkipRows>, <Direction>, ADOBM[#]);
```

Parameters

Parameter	Description
n	An index to the object. Criteria String value containing a statement that specifies the column name, comparison operator, and value to use in the search.
SkipRows	(Optional) Long value specifying the row offset from the current row or Start bookmark to begin the search. Default is zero. The search starts on the current row, by default.
SearchDirection	(Optional) SearchDirectionEnum value specifying if the search should begin on the current or next available row in the direction of the search. Valid values are: adSearchForward: unsuccessful search will stop at the end of the

	Recordset. adSearchBackward: unsuccessful search will stop at the start of the Recordset.
Start	(Optional) Variant bookmark that is the starting position for the search.

Example

```
ADO_Recordset(0)->AddNew( pvSource, pvValue );
EndCheckpoint("ADORecordset::AddNew");
BeginCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Find("test_number = 99", 0, adSearchForward, ADOBM[2] );
EndCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Delete( adAffectCurrent );
```

ADO_Recordset(n)->GetAbsolutePage

Identifies, by page number, where the current record resides.

Syntax

```
ADO_Recordset(n)->GetAbsolutePage( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long returning the page number.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetAbsolutePosition

Specifies the ordinal position of the current record of an ADO Recordset object. Use this method to locate a record based on its ordinal position, or to determine the current record's ordinal position. This is only available if your provider supports the appropriate functionality.

Syntax

```
ADO_Recordset(n)->GetAbsolutePosition( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetActiveCommand

Specifies the ADO Command object which created an ADO Recordset object. A Null object reference is returned if the ADO Recordset was not created by an ADO Command object.

Use this property to determine the ADO Command object when only the ADO Recordset object is known.

This function is only converted if there is an ADO Command object associated with this ADO Recordset. This is determined at conversion time.

Syntax

```
ADO_Recordset(n)->GetActiveCommand( ADOCommand[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOCommand	The ADO Command instance that is tied to this ADO Recordset.

ADO_Recordset(n)->GetActiveConnection

For a Command, ADO Recordset, or ADO Record object, specifies the associated ADO Connect object.

This property is read-only for open ADO Recordset objects or those whose Source property is set to a valid Command object. Otherwise, it is read/write.

Syntax

```
ADO_Recordset(n)->GetActiveConnection( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	Pointer to the Connection object.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetBOF

Determines if an ADO Recordset object contains records or if you've gone beyond its limits while moving from record to record.

If the current record position is before the first record, GetBOF returns True (-1). If it is on or after the first record, GetBOF returns False (0).

Syntax

```
ADO_Recordset(n)->GetBOF( pVT_BOOL );
```

Parameters

Parameter	Description
n	An index to the object.
pPT_BOOL	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->GetBookmark

Indicates a bookmark identifying an ADO Recordset object's current record, or sets the current record to that identified by a bookmark.

Use to save the position of the current record and return to it at any time. Bookmarks are available only in ADO Recordset objects that support bookmark functionality.

Syntax

```
ADO_Recordset(n)->GetBookmark( ADOBM[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOBM[#]	A Pointer to a CLoadBookmark instance.

Example

```
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->GetBookmark( ADOBM[0] );
```

ADO_Recordset(n)->GetCacheSize

Specifies the number of records in the ADO Recordset that are cached locally. Use to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory.

Syntax

```
ADO_Recordset(n)->GetCacheSize( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long value of the number of records in the ADO Recordset cached locally.

Example

```
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 6 );
```

ADO_Recordset(n)->GetCollect

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

 **Note:** Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->GetCollect( pvValue, pvData );
```

Parameters

Parameter	Description
n	An index to the object.
PvValue	A Pointer to a variant — perhaps the field name or ordinal.
PvData	A Pointer to a variant— perhaps the data for that field.

Example

```
ADO_Recordset(5)->GetState( pLong );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
```

ADO_Recordset(n)->GetCursorLocation

Specifies the library that the cursor service uses.

Allows you to choose between various cursor libraries accessible to the provider. Normally, the library can be client-side or on the server.

Syntax

```
ADO_Recordset(n)->GetCursorLocation( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 1 );
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseClient );
```

ADO_Recordset(n)->GetCursorType

Specifies the type of cursor to use when opening the ADO Recordset object. If the CursorLocation property is set to adUseClient, the only setting supported is adOpenStatic. If an unsupported value is set, the closest supported CursorType will be used instead.

Syntax

```
ADO_Recordset(n)->GetCursorType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenDynamic );
```

ADO_Recordset(n)->GetDataMember

Specifies the data member to be retrieved from the object referenced by the DataSource property. Creates data-bound controls with the Data Environment.

Syntax

```
ADO_Recordset(n)->GetDataMember( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.

sLoadStr	A CString containing a string representation of the data Member.
----------	--

Example

```
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenKeyset, adLockOptimistic, -1 );
ADO_Recordset(0)->GetDataMember( sLoadStr );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
```

ADO_Recordset(n)->GetDataSource

Specifies an object containing data to be represented as an ADO Recordset object. Creates data-bound controls with the Data Environment. GetDataSource takes a handle to an IUnknown as its argument. This is a pointer to a pointer. Please be careful dereferencing this element.

Syntax

```
ADO_Recordset(n)->GetDataSource( &pIUnknown );
```

Parameters

Parameter	Description
n	An index to the object.
&pIUnknown	A pointer to a pointer to a returned COM object.

Example

```
ADO_Recordset(1)->GetDataSource( &pIUnknown );
ADO_Recordset(2)->GetActiveConnection( pvValue );
ADO_Recordset(2)->GetCursorType( pLong );
```

ADO_Recordset(n)->GetEditMode

Specifies the current record's editing status.

Indicates whether changes have been made to this buffer associated with the current record, or whether a new record has been created. Use to determine the current record's editing status.

Syntax

```
ADO_Recordset(n)->GetEditMode( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```


ADO_Recordset(n)->GetEOF

Indicates that the current record position is after the last record in an ADO Recordset object. Returns True (-1) if the current record position is after the last record and False (0) if it is on or before the last record.

Syntax

```
ADO_Recordset(n)->GetEOF( pVTBOOL );
```

Parameters

Parameter	Description
n	An index to the object.
pVTBOOL	True (-1) or false (0).

Example


```
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->GetFields

Returns a container of an ADO Recordset or ADO Record object's Field objects.

QALoad's implementation of the GetFields method takes care of making the call through to the GetFields method within the ADO Recordset object. The Argument is one of the ADOFieldSet elements.

Retrieves an ADO Recordset object's ADO FieldSet object. This is an important step in variablization.

 **Note:** This function is not currently being converted in the script; however, this method can be used in conjunction with ADO_Field(n)->GetItem() to return data from a specific field of a particular recordset. It can be turned on or off using the QALoad Script Development Workbench's Convert Options wizard.

Syntax

```
ADO_Recordset(n)->GetFields( ADOFieldSet[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOFieldSet	An instance of the container of fields for a particular recordset.

Example

The following example illustrates returning the first field from the current recordset.

```
ADO_LoadVariant( pvSource, "8", "select * from test_table" );
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;
dsn=" FhLoadDB2;uid=sa;pwd=:database=Master;" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenDynamic, adLockPessimistic, 1 );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
ADO_FieldSet(0)->Refresh();
```

```
ADO_LoadVariant( pvValue, "2", "1" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
```

ADO_Recordset(n)->GetFilter

Specifies a filter for data in an ADO Recordset.

Use to screen out records in an ADO Recordset object. The filtered ADO Recordset becomes the current cursor.

Syntax

```
ADO_Recordset(n)->GetFilter( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```

ADO_Recordset(n)->GetIndex

This is a hidden method. It is undocumented within MSDN.

 **Note:** Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->GetIndex( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString object encapsulating some string data.

ADO_Recordset(n)->GetLockType

Specifies the type of locks placed on records during editing.

Set before opening an ADO Recordset to determine what type of locking the provider should use when opening the ADO Recordset. Read the property to return the type of locking in us.

Syntax

```
ADO_Recordset(n)->GetLockType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenForwardOnly );
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockOptimistic );
```

ADO_Recordset(n)->GetMarshalOptions

Specifies records to be marshaled back to the server.

Syntax

```
ADO_Recordset(n)->GetMarshalOptions( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long.

Example

```
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
ADO_Recordset(0)->GetMarshalOptions( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
```

ADO_Recordset(n)->GetMaxRecords

Specifies the maximum number of records to return to an ADO Recordset from a query.

Use to limit the number of records that the provider returns. The default, zero, indicates the provider will return all requested records.

Syntax

```
ADO_Recordset(n)->GetMaxRecords( pLong );
```

Parameters

Parameters	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockReadOnly );
```

QALoad 5.02

```
ADO_Recordset(0)->GetMaxRecords( pLong );  
ADO_Recordset(0)->PutMaxRecords( 10 );
```

ADO_Recordset(n)->GetPageCount

Specifies the number of pages of data contained in the ADO Recordset object.

Syntax

```
ADO_Recordset(n)->GetPageCount( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->PutPageSize( 4 );  
ADO_Recordset(0)->GetPageCount( pLong );  
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->GetPageSize

Indicates the number of records that make up a single page in the ADO Recordset.

Use to determine how many records make up a logical page of data, which allows you to use the AbsolutePage property.

Syntax

```
ADO_Recordset(n)->GetPageSize( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->GetMarshalOptions( pLong );  
ADO_Recordset(0)->GetPageSize( pLong );  
ADO_Recordset(0)->PutPageSize( 4 );  
ADO_Recordset(0)->GetPageCount( pLong );  
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->GetProperties

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

Retrieves the complete set of properties for this particular instance of the Recordset object. Property sets may change for different providers.

Syntax

```
ADO_Recordset(n)->GetProperties( CAPropertySet* pPropertySet );
```

Parameters

Parameter	Description
n	An index to the object.
pPropertySet	Set of CAProperty objects. Each CAProperty object contains a single characteristic, a piece of data, which partially describes the state of a particular instance of an object.

Example

```
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->GetState( pLong );
ADO_Recordset(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Recordset(n)->GetRecordCount

Indicates the number of records in an ADO Recordset object.

Use to determine how many records are in an ADO Recordset object. If ADO cannot determine the number, or if the provider or cursor type doesn't support RecordCount, GetRecordCount returns -1. An error results if GetRecordCount is used on a closed ADO Recordset.

Syntax

```
ADO_Recordset(n)->GetRecordCount( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetRows

Retrieves multiple records of an ADO Recordset object into an array.

Use the GetRows method to copy records from an ADO Recordset into a two-dimensional array. The first subscript identifies the field and the second identifies the record number. The array variable is automatically dimensioned to the correct size when the GetRows method returns the data.

Syntax

```
ADO_Recordset(n)->GetRows( #, pvSource, pvData, pvValue );
```

Or

```
ADO_Recordset(n)->GetRows( #, ADOBM[#], pvData, pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a variant containing the starting point for returning the data.
ADOBM[#]	A CLoadBookmark giving the starting point for the row retrieval.
pvData	A pointer to a variant containing an Array of fields to return.
pvValue	A Pointer to a variant containing a SafeArray into which the data is returned.

Example

```
ADO_LoadVariant( pvSource, "8", "select sID, sTable, sField, sPermits from" "Permits where  
sid = (select suserid from" "User where suserid = 'admin') or sid " "in (select sgroupid  
from UserGroup where suserid = 'admin') " "order by sPermits desc, sTable asc" );  
LoadVariant(pvValue, ADOConnect[1] );  
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenDynamic, adLockReadOnly, -1 );  
ADO_Recordset(0)->GetRecordCount( pLong );  
ADO_LoadVariant( pvSource, "10", "2147614724" );  
ADO_LoadVariant( pvData, "10", "2147614724" );  
ADO_Recordset(0)->GetRows( -1, pvSource, pvData, pvValue );  
EndCheckpoint("ADORecordset::GetRows" );
```

ADO_Recordset(n)->GetSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

Syntax

```
ADO_Recordset(n)->GetSort( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString containing the field to sort by.

ADO_Recordset(n)->GetSource

Indicates the data source for a Recordset object.

Use the Source property to specify a data source for a Recordset object using one of the following: a Command object variable, an SQL statement, a stored procedure, or a table name.

The Variant that is passed into the function is initialized before the call is made so that it will properly receive the variant information coming back from the call.

Syntax

```
ADO_Recordset(n)->GetSource( pvSource );
```

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a VARIANT.

Example

```
ADO_LoadVariant( pvValue, "3", "0" );
ADO_Recordset(0)->PutFilter( pvValue );
ADO_Recordset(0)->GetSource( pvSource );
ADO_Recordset(0)->GetStatus( pLong );
```

ADO_Recordset(n)->GetState

Indicates for all applicable objects whether the state of the object is open or closed.

Indicates for all applicable objects executing an asynchronous method, whether the current state of the object is connecting, executing, or retrieving.

Syntax

```
ADO_Recordset(n)->GetState( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a long.

Example

```
ADO_Recordset(0)->PutMaxRecords( 0 );
ADO_Recordset(0)->GetState( pLong );
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
```

ADO_Recordset(n)->GetStatus

Indicates the status of the current record with respect to batch updates or other bulk operations.

Syntax

```
ADO_Recordset(n)->GetStatus( pLong );
```

Parameters

Parameter	Description
n	An index to the object.

pLong	A pointer to a long.
-------	----------------------

Example

```
ADO_LoadVariant( pvValue, "3", "0" );
ADO_Recordset(0)->PutFilter( pvValue );
ADO_Recordset(0)->GetSource( pvSource );
ADO_Recordset(0)->GetStatus( pLong );
```

ADO_Recordset(n)->GetStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

This property applies to hierarchical recordsets, such as those supported by the Microsoft Data Shaping Service for OLE DB, and must be set on the parent ADO Recordset before the child ADO Recordset is retrieved. This property simplifies navigating hierarchical recordsets.

Since the VARIANT_BOOL datatype used by ADO is a direct mapping to the short datatype, QALoad uses the short datatype for this call.

Syntax

```
ADO_Recordset(n)->GetStayInSync( pVTBOOL );
```

Parameters

Parameter	Description
n	An index to the object.
pVTBOOL	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
ADO_Recordset(0)->PutStayInSync( FALSE );
ADO_LoadVariant( pvSource, "8", "select * from test_table where keyval < 100" ;
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;dsn="
"FhLoadDB2;uid=sa;pwd=;database=Master;" );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenUnspecified, adLockUnspecified, -1 );
EndCheckpoint("ADORecordset::Open");
```

ADO_Recordset(n)->GetString

Returns the ADO Recordset as a string.

Row data, but no schema data, is saved to the string. Therefore, an ADO Recordset cannot be re-opened using this string.

Syntax

```
ADO_Recordset(n)->GetString( sLoadStr );
```

Parameters

Parameter	Description
-----------	-------------

n	An index to the object.
sLoadStr	A CLoadString containing the string being returned.

Example

```
ADO_Recordset(0)->MoveFirst();
BeginCheckpoint("ADORecordset::GetString");
ADO_Recordset(0)->GetString( adClipString, -1, "", "", "", sLoadStr );
EndCheckpoint("ADORecordset::GetString");
ADO_LoadVariant( pvValue, "3", "1" );
BeginCheckpoint("ADORecordset::Move");
ADO_Recordset(0)->Move( 5, pvValue );
EndCheckpoint("ADORecordset::Move");
```

ADO_Recordset(n)->Move

Moves the position of the current record in an ADO Recordset object.

If the NumRecords argument is greater than zero, the current record position moves forward (toward the end of the ADO Recordset). If NumRecords is less than zero, the current record position moves backward (toward the beginning of the ADO Recordset).

If the Move call would move the current record position to a point before the first record, ADO sets the current record to the position before the first record in the recordset (BOF is True). An attempt to move backward when the BOF property is already True generates an error.

Syntax

```
ADO_Recordset(n)->Move( #, pvValue );
```

or

```
ADO_Recordset(n)->Move( #, ADOBM[#] );
```

Parameters

Parameter	Description
n	An index to the object.
#	Specifies the number of records that the current record position moves.
pvValue	A String value evaluates to a bookmark serving as the starting point.
ADOBM[#]	ACLoadBookmark serving as the starting point.

Example

```
BeginCheckpoint("ADORecordset::GetString");
ADO_Recordset(0)->GetString( adClipString, -1, "", "", "", sLoadStr );
EndCheckpoint("ADORecordset::GetString");
ADO_LoadVariant( pvValue, "3", "1" );
BeginCheckpoint("ADORecordset::Move");
ADO_Recordset(0)->Move( 5, pvValue );
EndCheckpoint("ADORecordset::Move");
```

ADO_Recordset(n)->MoveFirst

Use the MoveFirst method to move the current record position to the first record in the ADO Recordset.

Syntax

```
ADO_Recordset(n)->MoveFirst();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * From test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->MoveLast

Use the MoveLast method to move the current record position to the last record in the ADO Recordset. The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

Syntax

```
ADO_Recordset(n)->MoveLast();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
```

ADO_Recordset(n)->MoveNext

Use the MoveNext method to move the current record position one record forward (toward the bottom of the ADO Recordset). If the last record is the current record and you call the MoveNext method, ADO sets the current record to the position after the last record in the ADO Recordset (EOF is True). An attempt to move forward when the EOF property is already True generates an error.

Syntax

```
ADO_Recordset(n)->MoveNext( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * From test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->MovePrevious

Use the MovePrevious method to move the current record position one record backward (toward the top of the ADO Recordset). The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the MovePrevious method, ADO sets the current record to the position before the first record in the ADO Recordset (BOF is True).

Syntax

```
ADO_Recordset(n)->MovePrevious();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
```

ADO_Recordset(n)->NextRecordset

Clears the current ADO Recordset object and returns the next ADO Recordset by advancing through a series of commands.

Use the NextRecordset method to return the results of the next command in a compound command statement or of a stored procedure that returns multiple results. If you open an ADO Recordset object based on a compound command statement (for example, "SELECT * FROM table1;SELECT * FROM table2") using the Execute method on a Command or the Open method on an ADO Recordset, ADO executes only the first command and returns the results to recordset.

Syntax

```
ADO_Recordset(n)->NextRecordset( pvValue, ADORecordset[#] );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.
ADORecordset	ADORecordset instantiated by the returned data.

ADO_Recordset(n)->Open

Using the Open method on an ADO Recordset object opens a cursor that represents records from a base table, the results of a query, or a previously saved ADO Recordset.

Syntax

```
ADO_Recordset(n)->Open( pvSource, pvValue, <CursorTypeEnum>, <LockTypeEnum>, # );
```

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a VARIANT.
pvValue	A pointer to a VARIANT.
CursorTypeEnum	The CursorTypeEnum argument can be any of several string type values <ad!^%&> or a simple numeric representation.
LockTypeEnum	This is the type of locking that should occur to the recordset while the cloning operation is taking place.
#	Either a CommandTypeEnum or an ExecuteOptionEnum.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * FROM Test_Table" );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADOREcordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADOREcordset::Open");
ADO_Recordset(0)->Close();
ADOREcordset.Release( 0, ADOBM );
```

ADO_Recordset(n)->PutAbsolutePage

Indicates on which page the current record resides.

Use the AbsolutePage property to identify the page number on which the current record of the ADO Recordset is located.

Syntax

```
ADO_Recordset(n)->PutAbsolutePage( <PositionEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
PositionEnum	Any one of the following values: adPosUnknown, adPosBOF, or adPosEOF.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
```

```
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->PutAbsolutePosition

Indicates the ordinal position of an ADO Recordset object's current record.

Use the AbsolutePosition property to move to a record based on its ordinal position in the ADO Recordset object, or to determine the ordinal position of the current record. The provider must support the appropriate functionality for this property to be available.

Syntax

```
ADO_Recordset(n)->PutAbsolutePosition( <PositionEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
PositionEnum	Any one of the following values: adPosUnknown, adPosBOF, or adPosEOF.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->PutActiveConnection

Indicates to which Connection object the specified Command, ADO Recordset, or Record object currently belongs.

For open ADO Recordset objects or for ADO Recordset objects whose Source property is set to a valid Command object, the ActiveConnection property is read-only. Otherwise, it is read/write.

Syntax

```
LoadVariant( pvValue, ADOConnect[#] );
ADO_Recordset(n)->PutActiveConnection( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
```

QALoad 5.02

```
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->PutBookmark

Indicates a bookmark that uniquely identifies the current record in an ADO Recordset object or sets the current record in an ADO Recordset object to the record identified by a valid bookmark.

Use the Bookmark property to save the position of the current record and return to that record at any time. Bookmarks are available only in ADO Recordset objects that support bookmark functionality.

Syntax

```
ADO_Recordset(n)->PutBookmark( ADOBM[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOBM[#]	A CLoadBookmark object containing a bookmark associated with the element.

Example

```
BeginCheckpoint( "ADORecordset::GetBookmark" );
ADO_Recordset(0)->GetBookmark( ADOBM[0] );
EndCheckpoint( "ADORecordset::GetBookmark" );
ADO_Recordset(0)->MoveLast();
BeginCheckpoint( "ADORecordset::PutBookmark" );
ADO_Recordset(0)->PutBookmark( ADOBM[0] );
EndCheckpoint( "ADORecordset::PutBookmark" );
```

ADO_Recordset(n)->PutCacheSize

Indicates the number of records in the ADO Recordset that are cached locally.

Use the CacheSize property to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory. For example, if the CacheSize is 10, after first opening the ADO Recordset object, the provider retrieves the first 10 records into local memory.

Syntax

```
ADO_Recordset(n)->PutCacheSize( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	The size of the cache, a positive integer.

Example

```
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
```

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 6 );
```

ADO_Recordset(n)->PutCollect

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit. Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->PutCollect( pvValue, pvSource );
```

Parameters

Parameter	Description
n	An index to the object.
PvValue	A Pointer to a variant — perhaps the field name or ordinal.
PvData	A Pointer to a variant — perhaps the data for that field.

Example

```
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_LoadVariant( pvData, "8", "333555333" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
ADO_LoadVariant( pvData, "8", "Gifford" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
ADO_LoadVariant( pvData, "8", "Roger" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_LoadVariant( pvData, "8", "X" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
```

ADO_Recordset(n)->PutCursorLocation

Indicates the location of the cursor service.

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

Syntax

```
ADO_Recordset(n)->PutCursorLocation( <CursorLocationEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
CursorLocationEnum	The place where the cursor is drawn from. Any one of the following: adUseNone, adUseServer, adUseClient, adUseClientBatch.

Example

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 1 );
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseClient );
```

ADO_Recordset(n)->PutCursorType

Use the CursorType property to specify the type of cursor that should be used when opening the ADO Recordset object.

Only a setting of adOpenStatic is supported if the CursorLocation property is set to adUseClient. If an unsupported value is set, then no error will result; the closest supported CursorType will be used instead.

Syntax

```
ADO_Recordset(n)->PutCursorType( <CursorTypeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
CursorTypeEnum	The CursorTypeEnum argument can be: adOpenUnspecified, adOpenForwardOnly, adOpenKeyset, adOpenDynamic, or adOpenStatic.

Example

```
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenDynamic );
```

ADO_Recordset(n)->PutDataMember

Indicates the name of the data member that will be retrieved from the object referenced by the DataSource property.

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (data members) that will be represented as an ADO Recordset object.

Syntax

```
ADO_Recordset(n)->PutDataMember( "<DataMember>" );
```

Parameters

Parameter	Description
n	An index to the object.
<DataMember>	Name of the data member being returned from the Recordset object.

ADO_Recordset(n)->PutFilter

Indicates a filter for data in an ADO Recordset.

Use the Filter property to selectively screen out records in an ADO Recordset object. The filtered ADO Recordset becomes the current cursor.

Syntax

```
ADO_Recordset(n)->PutFilter( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```

ADO_Recordset(n)->PutIndex

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->PutIndex( "Index" );
```


Parameters

Parameter	Description
n	An index to the object.
Index	The Index on the recordset.

ADO_Recordset(n)->PutLockType

Indicates the type of locks placed on records during editing.

Set the LockType property before opening an ADO Recordset to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open ADO Recordset object.

 **Note:** The LockTypeEnum argument can be any of several elements listed below, or it may be a cast number — (LockTypeEnum)0. For best results when load testing, please feel free to replace the lock type with adLockOptimistic.

```
adLockUnspecified
adLockReadOnly
adLockPessimistic
```

adLockOptimistic
adLockBatchOptimistic

Syntax

```
ADO_Recordset(n)->PutLockType( <LockTypeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
LockTypeEnum	An enumeration of lock types.

Example

```
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenForwardOnly );
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockOptimistic );
```

ADO_Recordset(n)->PutMarshalOptions

Indicates which records are to be marshaled back to the server.

Syntax

```
ADO_Recordset(n)->PutMarshalOptions( <MarshalOptionsEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
MarshalOptionsEnum	Indicator about records to send across.

Example

```
ADO_Recordset(2)->PutMarshalOptions( adMarshalModifiedOnly );
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_LoadVariant( pvData, "8", "333555333" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
ADO_LoadVariant( pvData, "8", "Gifford" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
ADO_LoadVariant( pvData, "8", "Roger" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_LoadVariant( pvData, "8", "X" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
```

ADO_Recordset(n)->PutMaxRecords

Indicates the maximum number of records to return to an ADO Recordset from a query.

Use the MaxRecords property to limit the number of ADO Records that the provider returns from the data source. The default setting of this property is zero, which means the provider returns all requested records.

Syntax

```
ADO_Recordset(n)->PutMaxRecords( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	Maximum number of records to return from a Recordset query.

Example

```
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockReadOnly );
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->PutMaxRecords( 10 );
```

ADO_Recordset(n)->PutPageSize

Indicates how many records constitute one page in the ADO Recordset.

Use the PageSize property to determine how many ADO Records make up a logical page of data. Establishing a page size allows you to use the AbsolutePage property to move to the first record of a particular page.

Syntax

```
ADO_Recordset(n)->PutPageSize( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	Number of records forming a page in the Recordset.

Example

```
ADO_Recordset(0)->GetMarshalOptions( pLong );
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageCount( pLong );
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->PutRefActiveConnection

Indicates to which ADO Connect object the specified ADO Command, ADO Recordset, or Record object currently belongs.

For open ADO Recordset objects or for ADO Recordset objects whose Source property is set to a valid ADO Command object, the ActiveConnection property is read-only. Otherwise, it is read/write.

Syntax

```
ADO_Recordset(n)->PutRefActiveConnection( ADOConnect[#] );
```

Parameters

Parameter	Description
n	An index to the object.
ADOCommand[#]	The connection that is active.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->PutRefDataSource

Indicates an object that contains data to be represented as an ADO Recordset object.

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (data members) that will be represented as an ADO Recordset object.

Syntax

```
ADO_Recordset(n)->PutRefDataSource( pIUnknown );
```

Parameters

Parameter	Description
n	An index to the object.
PIUnknown	A pointer to a COM object.

ADO_Recordset(n)->PutRefSource

Sets a Command object as the data source for a Recordset object.

Use the Source property to specify a data source for a Recordset object using one of the following: a Command object variable, SQL statement, stored procedure, or table name.

Syntax

```
ADO_Recordset(n)->PutRefSource( VARIANT* pvValue );
```

Parameters

Parameter	Description
pvValue	A pointer to a variant that contains a reference to a valid Command object.

Example

```
ADO_Recordset(0)->PutActiveConnection( pvValue );
LoadVariant( pvValue, ADOCommand[0] );
```

```
ADO_Recordset(0)->PutRefSource( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
```

ADO_Recordset(n)->PutSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

Syntax

```
ADO_Recordset(n)->PutSort( "SortString" );
```

Parameters

Parameter	Description
n	An index to the object.
SortString	char*

ADO_Recordset(n)->PutSource

Indicates the data source for an ADO Recordset object.

Use the Source property to specify a data source for an ADO Recordset object using one of the following: a Command object variable, an SQL statement, a stored procedure, or a table name.

Syntax

```
ADO_Recordset(n)->PutSource( "<DataSource>" );
```

Parameters

Parameter	Description
n	An index to the object.
<Datasource>	A char* representation of a data source.

Example

```
ADO_Recordset(0)->PutSource( "Select sUID, sPWD, sPhone" "from USER Where lcase(sUID)='sa'
and sPWD = 'sa'" );
ADO_LoadVariant( pvSource, "8", "Select sUID, sPWD, sPhone" "from USER Where
lcase(sUID)='sa' and sPWD = 'sa'" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenKeyset, adLockOptimistic, -1 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
```

ADO_Recordset(n)->PutStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

This property applies to hierarchical ADO Recordsets, such as those supported by the Microsoft Data Shaping Service for OLE DB, and must be set on the parent ADO Recordset before the child ADO Recordset is retrieved. This property simplifies navigating hierarchical ADO Recordsets.

Syntax

```
ADO_Recordset(n)->PutStayInSync( <BOOL> );
```

Parameters

Parameter	Description
n	An index to the object.
BOOL	TRUE or FALSE.

Example

```
ADO_Recordset(0)->GetStayInSync( FALSE );
ADO_Recordset(0)->PutStayInSync( pVTBOOL );
ADO_LoadVariant( pvSource, "8", "select * from test_table where keyval < 100" ;
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;dsn="
"FhLoadDB2;uid=sa;pwd=;database=Master;" );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenUnspecified, adLockUnspecified, -1 );
EndCheckpoint("ADORecordset::Open");
```

ADO_Recordset(n)->ReQuery

Updates the data in an ADO Recordset object by re-executing the query on which the object is based.

Use the Requery method to refresh the entire contents of an ADO Recordset object from the data source by reissuing the original command and retrieving the data a second time.

Syntax

```
ADO_Recordset(n)->ReQuery( # );
```

Parameters

Parameter	Description
n	An index to the object.
#	-1 TRUE or 0 FALSE

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
ADO_Recordset(0)->Close();
```

ADO_Recordset(n)->Resync

Refreshes the data in the current ADO Recordset object, or Fields collection of a Record object, from the underlying database.

Use the Resync method to resynchronize records in the current ADO Recordset with the underlying database. This is useful if you are using either a static or forward-only cursor, but you want to see any changes in the underlying database.

Syntax

```
ADO_Recordset(n)->Resync( <AffectEnum>, <ResyncEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
AffectEnum	An enumerated type , any one of the following values: adAffectCurrent, adAffectGroup, adAffectAll, adAffectAllChapters.
ResyncEnum	An enumerated type , any one of the following values: adResyncUnderlyingValues, adResyncAllValues.

Example


```
ADO_Recordset(0)->Resync(adAffectAll, adResyncAllValues );
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
```

ADO_Recordset(n)->Save

Saves the ADO Recordset in a file or ADO Stream object. The Save method can only be invoked on an open ADO Recordset. Use the Open method to later restore the ADO Recordset from Destination.

If the Filter property is in effect for the ADO Recordset, then only the rows accessible under the filter are saved. If the ADO Recordset is hierarchical, then the current child ADO Recordset and its children are saved, including the parent ADO Recordset. If the Save method of a child ADO Recordset is called, the child and all its children are saved, but the parent is not.

The first time you save the ADO Recordset, it is optional to specify Destination. If you omit Destination, a new file will be created with a name set to the value of the Source property of the ADO Recordset.

 **Note:** The second argument here can be given as adPersistXML or adPersistADTG.

Syntax

```
ADO_Recordset(n)->Save( pvValue, <PersistModeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.
PersistModeEnum	An enumerated type.

Example

```
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_LoadVariant( pvValue, "8", "saver.xml" );
ADO_Recordset(0)->Save( pvValue, adPersistXML );
```

ADO_Recordset(n)->Seek

The SeekEnum is an Enumerated value giving the direction of the seek operation.

Syntax

```
ADO_Recordset(n)->Seek( pvValue, <SeekEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a Variant containing an array of Variant values.
SeekEnum	Direction of the seek.

ADO_Recordset(n)->Supports

Determines whether a specified ADO Recordset object supports a particular type of functionality. If the ADO Recordset object supports the features whose corresponding constants are in CursorOptions, the Supports method returns True. Otherwise, it returns False.

Syntax

```
ADO_Recordset(n)->Supports( CursorOptionEnum, pVT_BOOL );
```

Parameters

Parameter	Description
n	An index to the object.
CursorOptionEnum	An enumerated type.
pVT_BOOL	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
ADO_Recordset(0)->Close();
```

ADO_Recordset(n)->Update

Saves any changes you make to the current row of an ADO Recordset object.

Use the Update method to save any changes you make to the current record of an ADO Recordset object since calling the AddNew method or since changing any field values in an existing record. The ADO Recordset object must support updates.

Syntax

```
ADO_Recordset(n)->Update( pvValue, pvData );
```


Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.
pvData	A pointer to a VARIANT.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "FirstName" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "3", "John" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "LastName" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Doe" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
```

ADO_Recordset(n)->UpdateBatch

Writes all pending batch updates within the ADO Recordset to disk.

Syntax

```
ADO_Recordset(n)->UpdateBatch( <AffectEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
AffectEnum	An enumerated type.

Example

```
ADO_Recordset(0)->Delete( adAffectCurrent );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
BeginCheckpoint("ADORecordset::Update");
ADO_Recordset(0)->Update( pvValue, pvData );
EndCheckpoint("ADORecordset::Update");
BeginCheckpoint("ADORecordset::UpdateBatch");
ADO_Recordset(0)->UpdateBatch( adAffectAll );
EndCheckpoint("ADORecordset::UpdateBatch");
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
```

ADO_Stream(n)->Cancel

Cancels execution of a pending ADO Stream, asynchronous method call.

Syntax

```
ADO_Stream(n)->Cancel();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open(pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->Close

Closes an open object and any dependent objects.

Using the Close method to close an ADO Stream object releases the associated data and any exclusive access you may have had to the data through this particular object. You can later call the Open method to reopen the object with the same, or modified, attributes.

Close the ADO Stream and give up all rights you may have had to the data.

Syntax

```
ADO_Stream(n)->Close();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->CopyTo

Copies the specified number of characters or bytes (depending on Type) in the ADO Stream to another ADO Stream object.

This method copies the specified number of characters or bytes, starting from the current position specified by the Position property. If the specified number is more than the available number of bytes until EOS, then only characters or bytes from the current position to EOS are copied. If the value of NumChars is 1, or omitted, all characters or bytes starting from the current position are copied.

If there are existing characters or bytes in the destination ADO Stream, all contents beyond the point

where the copy ends remain, and are not truncated. Position becomes the byte immediately following the last byte copied. If you want to truncate these bytes, call `SetEOS`.

Syntax

```
ADO_Stream(n)->CopyTo( ADOStream[ # ], # );
```

Parameters

Parameter	Description
n	An index to the object.
ADOStream[#]	An instance of an ADO Source object.
#	A positive integer.

Example

```
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->CopyTo( ADOStream[1], 5 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
```

ADO_Stream(n)->Flush

Forces the contents of the ADO Stream remaining in the ADO buffer to the underlying object with which the ADO Stream is associated.

This method may be used to send the contents of the ADO Stream buffer to the underlying object represented by the URL that is the source of the ADO Stream object. This method should be called when you want to ensure that all changes made to the contents of an ADO Stream have been written. However, with ADO it is not usually necessary to call `Flush`, as ADO continuously flushes its buffer as much as possible in the background.

Changes to the content of an ADO Stream are made automatically, and not cached until `Flush` is called.

Syntax

```
ADO_Stream(n)->Flush( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(0)->CopyTo( ADOStream[1], 5 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->Flush();
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->GetCharset

Indicates the character set into which the contents of a text ADO Stream should be translated.

In a text ADO Stream object, text data is stored as Unicode. The Charset property translates the data read from the ADO Stream into the specified character set. Similarly, data written to the ADO Stream in the specified character set is translated into Unicode for storage in the ADO Stream object.

Syntax

```
ADO_Stream(n)->GetCharset( sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
sLoadStr	A CLoadString object.

Example

```
ADO_Stream(1)->Cancel();
ADO_Stream(1)->Close();
ADO_Stream(0)->GetCharset( sLoadStr );
ADO_Stream(0)->PutCharset( "Unicode" );
```

ADO_Stream(n)->GetEOS

Indicates whether the current position is at the end of the ADO Stream.

Returns a Boolean value that indicates whether the current position is at the end of the ADO Stream. EOS returns True if there are no more bytes in the ADO Stream; it returns False if there are more bytes following the current position.

At replay, QALoad checks the current position in the stream to determine whether or not this is the end of the stream.

Syntax

```
ADO_Stream(n)->GetEOS( pVTBOOL );
```

Parameters

Parameter	Description
n	An index to the object.
pVT_BOOL	A pointer to a VARIANT_BOOL.

Example

```
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->GetLineSeparator

Indicates the binary character to be used as the line separator in text ADO Stream objects.

LineSeparator is used only with text ADO Stream objects (Type is adTypeText). This property is ignored if Type is adTypeBinary.

Syntax

```
ADO_Stream(n)->GetLineSeparator( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
ADO_Stream(1)->GetLineSeparator( pLong );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->GetMode

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

Use the Mode property to set or return the access permissions in use by the provider on the current connection. You can set the Mode property only when the Connection object is closed.

Syntax

```
ADO_Stream(n)->GetMode( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetLineSeparator( pLong );
ADO_Stream(0)->PutLineSeparator( adCRLF );
ADO_Stream(0)->GetState( pLong );
ADO_Stream(0)->GetMode( pLong );
ADO_Stream(0)->PutMode( adModeShareDenyNone );
```

ADO_Stream(n)->GetPosition

Indicates the current position within an ADO Stream object.

Sets or returns a Long value that specifies the offset, in number of bytes, of the current position from the beginning of the ADO Stream. The default is 0, which represents the first byte in the ADO Stream.

At replay, QALoad, checks the current position and feeds that position back to the user in the pointer.

Syntax

```
ADO_Stream(n)->GetPosition( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
```

ADO_Stream(n)->GetSize

Returns a Long value that specifies the size of the ADO Stream in number of bytes. The default value is the size of the ADO Stream, or -1 if the size of the ADO Stream is not known.

At replay, QALoad checks the size of the ADO Stream that is referenced by this call.

Syntax

```
ADO_Stream(n)->GetSize( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->PutPosition( 0 );
```

ADO_Stream(n)->GetState

The ADO Stream object's State property can have a combination of values. For example, if a statement is executing, this property will have a combined value of adStateOpen and adStateExecuting.

GetState returns 0 for a not open state and 1 for an open state.

Syntax

```
ADO_Stream(n)->GetState( pLong );
```

Parameters

Parameter	Description
-----------	-------------

n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```

ADO_Stream(n)->GetType

Indicates the type of data contained in the ADO Stream (binary or text).

Sets or returns a StreamTypeEnum value that specifies the type of data contained in the ADO Stream object. The default value is adTypeText. However, if binary data is initially written to a new, empty ADO Stream, the Type will be changed to adTypeBinary.

Syntax

```
ADO_Stream(n)->GetType( pLong );
```

Parameters

Parameter	Description
n	An index to the object.
pLong	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)->GetType( pLong );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
```

ADO_Stream(n)->LoadFromFile

Loads the contents of an existing file into an ADO Stream.

This method may be used to load the contents of a local file into an ADO Stream object. This may be used to upload the contents of a local file to a server.

The ADO Stream object must be already open before calling LoadFromFile. This method does not change the binding of the ADO Stream object; it will still be bound to the object specified by the URL with which the ADO Stream was originally opened. LoadFromFile overwrites the current contents of the ADO Stream object with data read from the file.

Syntax

```
ADO_Stream(n)->LoadFromFile( "filename" );
```

Parameters

Parameter	Description
n	An index to the object.

<FileName>

A string representation of a file name.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```

ADO_Stream(n)->Open

Opens an ADO Stream object to manipulate streams of binary or text data.

When a Record object is passed in as the source parameter, the User ID and Password parameters are not used because access to the Record object is already available. Similarly, the Mode of the Record object is transferred to the ADO Stream object.

When Source is not specified, the ADO Stream opened contains no data and has a Size of zero (0). To avoid losing any data that is written to this ADO Stream when the ADO Stream is closed, save the ADO Stream with the CopyTo or SaveToFile methods, or save it to another memory location.

While the ADO Stream is not open, it is possible to read all the read-only properties of the ADO Stream. If an ADO Stream is opened asynchronously, all subsequent operations (other than checking the State and other read-only properties) are blocked until the Open operation is completed.

Open the ADO Stream to manipulate binary or text data.

Syntax

```
ADO_LoadVariant( pvValue, "Type", "Value" );
ADOStream(n)->Open( pvSource, <ConnectionModeEnum>,
    <StreamOpenOptionsEnum>, "User", "Password" );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.
ConnectionModeEnum	An enumerated data set.
UserName	A user name string.
Password	A password string.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```


ADO_Stream(n)->PutCharset

Indicates the character set into which the contents of a text ADO Stream should be translated.

In a text ADO Stream object, text data is stored as Unicode. The Charset property translates the data read from the ADO Stream into the specified character set. Similarly, data written to the ADO Stream in the specified character set is translated into Unicode for storage in the ADO Stream object.

Syntax

```
ADO_Stream(n)->PutCharset( "charset" );
```

Parameters

Parameter	Description
n	An index to the object.
Charset	A string representation of a character set.

Example

```
ADO_Stream(0)->PutCharset( "ascii" );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState(pLong );
ADO_Stream(0)->PutPosition( 15 );
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)->GetType( pLong );
```

ADO_Stream(n)->PutLineSeparator

Indicates the binary character to be used as the line separator in text ADO Stream objects.

LineSeparator is used only with text ADO Stream objects (Type is adTypeText). This property is ignored if Type is adTypeBinary.

Syntax

```
ADO_Stream(n)->PutLineSeparator( LineSeparatorEnum );
```

Parameters


Parameter	Description
n	An index to the object.
LineSeparatorEnum	Text or binary.

Example

```
ADO_Stream(0)->CopyTo( ADOSTream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
```

ADO_Stream(n)->PutMode

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object. Use the Mode property to set or return the access permissions in use by the provider on the current connection.

 **Note:** You can set the Mode property only when the Connection object is closed.

Syntax

```
ADO_Stream(n)->PutMode( <ConnectModeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
ADO_Stream(0)->PutMode( adModeShareDenyNone );
ADO_Stream(0)->PutPosition( 15 );
```

ADO_Stream(n)->PutPosition

Indicates the current position within an ADO Stream object.

Sets or returns a Long value that specifies the offset, in number of bytes, of the current position from the beginning of the ADO Stream. The default is 0, which represents the first byte in the ADO Stream.

At replay, QALoad sets the current position in the ADO Stream.

Syntax

```
ADO_Stream(n)->PutPosition( # );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)->GetType( pLong );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
```

ADO_Stream(n)->PutType

Indicates the type of data contained in the ADO Stream (binary or text).

Sets or returns a StreamTypeEnum value that specifies the type of data contained in the ADO Stream object. The default value is adTypeText. However, if binary data is initially written to a new, empty ADO Stream, the Type will be changed to adTypeBinary.

Syntax

```
ADO_Stream(n)->PutType( <StreamTypeEnum> );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->PutCharset( "ascii" );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
```

ADO_Stream(n)->Read

Reads a specified number of bytes from a binary ADO Stream object.

If NumBytes is more than the number of bytes left in the ADO Stream, only the bytes remaining are returned. The data read is not padded to match the length specified by NumBytes. If there are no bytes left to read, a variant with a null value is returned. Read cannot be used to read backwards.

Syntax

```
ADO_Stream(n)->Read( <NumBytes>, sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.
NumBytes	A non negative integer value corresponding to the number of bytes to read.
sLoadStr	A CLoadString into which the data is read.

ADO_Stream(n)->ReadText

Reads specified number of characters from a text ADO Stream object.

If NumChar is more than the number of characters left in the ADO Stream, only the characters remaining are returned. The string read is not padded to match the length specified by NumChar. If there are no characters left to read, a variant whose value is null is returned.

ReadText cannot be used to read backwards.

Syntax

```
ADO_Stream(n)->ReadText( <NumChar>, sLoadStr );
```

Parameters

Parameter	Description
n	An index to the object.

NumChar	A non negative integer value corresponding to the number of characters to read.
sLoadStr	A CLoadString into which the data is read.

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
```

ADO_Stream(n)->SaveToFile

Saves the number of bytes contents of the current ADO Stream to the file from the current position. It sends the second param number of bytes to that File.

SaveToFile may be used to copy the contents of an ADO Stream object to a local file. There is no change in the contents or properties of the ADO Stream object. The ADO Stream object must be open before calling SaveToFile.

This method does not change the association of the ADO Stream object to its underlying source. The ADO Stream object will still be associated with the original URL that was its source when opened.

Syntax

```
ADO_Stream(n)->SaveToFile( "FileName", <SaveOptionsEnum> );
```

Parameters

Parameter	Description
n	An index to the object.
FileName	A String forming the name of the file to save the stream to.
SaveOptionsEnum	Create or overwrite are the different options.

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
ADO_Stream(1)->GetLineSeparator( pLong );
ADO_Stream(1)->SaveToFile( "D:\\StreamReceive.txt", adSaveCreateOverWrite );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->SetEOS

Sets the current position within the ADO Stream as the End of the ADO Stream. SetEOS updates the value of the EOSproperty, by making the current Position the end of the ADO Stream. Any bytes or characters following the current position are truncated.

Syntax

```
ADO_Stream(n)->SetEOS( );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(1)->SaveToFile( "D:\\Streamward.txt", adSaveCreateOverWrite );
ADO_Stream(1)->GetPosition( pLong );
if( *pLong >=20 )
  ADO_Stream(1)->SetEOS();
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->SkipLine

Skips one entire line when reading a text ADO Stream.

All characters up to, and including the next line separator, are skipped. By default, the LineSeparator is adCRLF. If you attempt to skip past EOS, the current position will simply remain at EOS.

Skip a line in the text buffer that is the ADO Stream.

Syntax

```
ADO_Stream(n)->SkipLine();
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOSTream[1], 150 );
ADO_Stream(0)->ReadText( 50, sLoadStr );
ADO_Stream(0)->SkipLine();
ADO_Stream(1)->ReadText( 50, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
```

ADO_Stream(n)->Write

Write writes BINARY Data to the ADO Stream buffer. Specified bytes are written to the ADO Stream object without any intervening spaces between each byte.

The current Position is set to the byte following the written data. The Write method does not truncate the rest of the data in a stream. If you want to truncate these bytes, call SetEOS.

If you write past the current EOS position, the Size of the ADO Stream will be increased to contain any new bytes, and EOS will move to the new last byte in the ADO Stream.

Syntax

```
ADO_LoadVariant( pvValue, "type", "value" );
ADO_Stream(n)->Write( pvValue );
```

Parameters

Parameter	Description
n	An index to the object.

ADO_Stream(n)->WriteText

Writes a specified text string to an ADO Stream object.

Specified strings are written to the ADO Stream object without any intervening spaces or characters between each string.

The current Position is set to the character following the written data. The WriteText method does not truncate the rest of the data in a stream. If you want to truncate these characters, call SetEOS.

Syntax

```
ADO_Stream(n)->WriteText( "texttowrite", <StreamWriteEnum> );
```

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(1)->GetSize( pLong );
ADO_Stream(1)->PutPosition( *pLong );
ADO_Stream(1)->SetEOS();
ADO_Stream(1)->WriteText( "This is the way we drink water", adWriteLine );
```

ExtractVariantValue

Retrieves the contents of a variant and places that value in a string.

The resulting string can be used to provide users easier access to variant data. To prevent memory leaks, free the string after use with the free() method.

Syntax

```
char* ExtractVariantValue ( VARIANT* pValue );
```

Parameters

Parameter	Description
pValue	The variant from the script

Example

```
// extract value from a field
ADO_Field(0)->GetValue( pvValue );
// use that value to create a new query
char* pszVal = ExtractVariantValue( pvValue );
char* pszSelect = (char*)_malloc( 128 );
sprintf( pszSelect, "select * from test_table where keyval = '%s'", pszVal );
// clean up to avoid memory leaks
free( pszVal );
```

```
// open a new recordset
ADO_LoadVariant( pvSource, "8", pszSelect );
free( pszSelect );
ADO_LoadVariant( pvValue, "8",
"PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa;pwd=;database="Master" );
ADO_Recordset(4)->Open( pvSource, pvValue, adOpenUnspecified, adLockUnspecified, -1 );
```

PrintVariant

Decodes variant data and places this data into a string.

The resulting string is combined with the specified comment and printed to the Player window during a load test.

Syntax

```
void PrintVariant ( VARIANT* pValue, char* sComment );
```

Parameters

Parameter	Description
pValue	The variant from the script.
sComment	Comment to send to the Player window.

Example

```
// extract the value in the first column
ADO_Recordset(3)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "3", "0" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetValue( pvValue );
// print that value
PrintVariant( pvValue, "Value of the item in the first column" );
```

Citrix

Citrix Index

BeginBlock

End of an if block of code.

CitrixInit

Initializes Citrix replay middleware resources.

CitrixUninit

Un-initializes the Citrix replay middleware resources. If the connection is still open, the function disconnects it.

CTX_error_handler

Outputs a fatal error message to the Conductor, which causes the virtual user to either fail or report a warning.

CtxClick

Clicks the specified button, using the specified modifier, at the current location.

CtxConnect

Connects to the Citrix server with the specified hostname and output mode.

[CtxDisconnect](#)

Disconnects from the Citrix server.

[CtxDomainLoginInfo](#)

Connects to the Citrix server with the specified user name, password, and domain.

[CtxDoubleClick](#)

Double-clicks the mouse at the current location.

[CtxKeyDown](#)

Inputs the keystroke specified by the key argument, which corresponds to the VK code.

[CtxKeyUp](#)

Inputs the keystroke specified by the key argument, which corresponds to the VK code.

[CtxMouseDown](#)

Presses the specified mouse button.

[CtxMouseMove](#)

Moves the mouse to the given coordinates with the given button and modifier.

[CtxMouseUp](#)

Releases the specified mouse button.

[CtxPing](#)

Sends a ping request and waits for the response.

[CtxPoint](#)

Moves the mouse to the specified location on the screen.

[CtxScreenEventExists](#)

Waits for the specified screen update to occur at the specified coordinates.

[CtxSetApplication](#)

Connects to the Citrix server with the specified application name and application working directory name.

[CtxSetCitrixPort](#)

Sets the port for the Citrix client to use to connect to the server.

[CtxSetConnectTimeout](#)

Sets the number of seconds to wait for connections to the server to complete.

[CtxSetDisconnectTimeout](#)

Sets the number of seconds to wait for disconnections from the server to complete.

[CtxSetEnableCounters](#)

Enables or disables custom counters for Citrix client-side statistics.

[CtxSetEnableWildcardMatching](#)

Enables or disables wildcard and substring name comparisons for matching Citrix window creation events.

[CtxSetICAFile](#)

Uses the specified ICA file when connecting to a published application or desktop.

[CtxSetLoginInfo](#)

Connects to the Citrix server with the specified user name and password.

[CtxSetPingTimeout](#)

Sets the number of seconds to wait for a ping to be acknowledged.

[CtxSetWaitPointTimeout](#)

Sets the number of seconds to wait for a wait point.

[CtxSetWindowMatchTitle](#)

Sets the string to match the names of previously-created windows.

CtxSetWindowRetries

Sets the retry information for window verification.

CtxSetWindowTimeout

Sets the number of seconds to wait for windows to be activated and destroyed.

CtxSetWindowVerification

Enables or disables window verification for actions.

CtxType

Inputs the specified key strokes.

CtxTypeChar

Sends the specified ASCII character to the Citrix server.

CtxTypeVK

Sends the VK code that corresponds to a key typed by the user.

CtxWaitForCaptionChange

Waits for the specified window's caption to be changed.

CtxWaitForScreenUpdate

Waits for the specified screen update to occur at the specified coordinates.

CtxWaitForWindowActive

Waits for the specified window to be activated (brought to the foreground).

CtxWaitForWindowCreate

Waits for the specified window to be created.

CtxWaitForWindowDestroy

Waits for the specified window to be destroyed.

CtxWaitForWindowLglIconChange

Waits for the specified window's caption to be changed.

CtxWaitForWindowMinimize

Waits for the specified window to be minimized.

CtxWaitForWindowMove

Waits for the specified window to be moved to the specified coordinates.

CtxWaitForWindowResize

Waits for the specified window to be resized to the specified dimensions.

CtxWaitForWindowSmIconChange

Waits for the specified window's caption to be changed.

CtxWaitForWindowStyleChange

Waits for the specified window's style to be changed as specified.

CtxWindowEventExists

Checks to see if the specified window event has already occurred, and, if not, waits for the specified time for the event to occur.

EndBlock

End of an else block of code.

BeginBlock

End of an if block of code.

Syntax

```
void BeginBlock();
```

Return Value

None

Parameters

None

Example

```
// Window CWI_5 ("Citrix License Warning Notice") created 1087837373.062
if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE, 3000, CWI_5))
BeginBlock();
CtxWaitForWindowCreate(CWI_5, 46);
EndBlock();
```

CitrixInit

Initializes the Citrix replay middleware resources.

Syntax

```
void CitrixInit (int flags);
```

Return Value

None

Parameters

Parameter	Description
flags	Reserved

Example

```
CitrixInit(2);
```

CitrixUninit

Un-initializes the Citrix replay middleware resources. If the connection is still open, this function disconnects it.

Syntax

```
void CitrixUninit();
```

Return Value

None

Parameters

None

Example

```
CitrixUninit();
```

CTX_error_handler

Outputs a fatal error message to the Conductor, which causes the virtual user to either fail or report a warning.

Syntax

```
void CTX_error_handler(PPLAYERINFO *, char *msg);
```

Parameters

Parameter	Description
PLAYERINFO	Pointer to the PLAYERINFO struct, sinfo.
msg	Message to be passed to the Conductor.

Example

```
{
    char buffer[1024];
    sprintf(buffer, "App did not start. Stop script now!");
    CTX_error_handler(s_info, buffer);
}
```

CtxClick

Clicks the specified button, using the specified modifier, at the current location.

Syntax

```
void CtxClick(const CtxWI* windowInfo, long holdTime, long button, long ModifierKeys);
```

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
holdTime	Number of milliseconds to hold down the button.
button	A mouse button to use for this action. NONE: No mouse button was specified L_BUTTON: The left mouse button R_BUTTON: The right mouse button M_BUTTON: The middle mouse button
ModifierKeys	A keyboard modifier to use for this action. NONE: No keyboard modifier was specified SHIFT: Shift key CONTROL: Control key ALT: Alt key EXTENDED: An extended key

Example

```
CtxWI *CWI_7001c = new CtxWI(0x1001c, "Warning !!", 299, 139, 427, 351);
...
CtxClick(CWI_7001c, 109, L_BUTTON, NONE);
```

CtxConnect

Connects to the Citrix server with the specified hostname and output mode.

Syntax

```
void CtxConnect (const char *hostname, int outputmode);
```

Parameters

Parameter	Description
*hostname	Name of the server to connect to.
outputmode	Type of playback output/display. The following modes are available: OUTPUT_MODE_NORMAL: Seamless rendering and display with window management. OUTPUT_MODE_WINDOWLESS: Graphics are not displayed. OUTPUT_MODE_RENDERLESS: Graphics are not used or displayed and there is no window management.

Example

```
const char *CitrixServer      = "qaccitrix";
const int   CitrixOutputMode = OUTPUT_MODE_NORMAL;
...
CtxConnect(CitrixServer, CitrixOutputMode);
```

CtxDisconnect

Disconnects from the Citrix server.

Syntax

```
void CtxDisconnect();
```

Return Value

None

Parameters

None

Example

```
CtxDisconnect();
```

CtxDoubleClick

Double-clicks the mouse at the current location. If Window Verification is enabled, ensure that the specified window is in the foreground.

Syntax

```
void CtxDoubleClick(const CtxWI* windowInfo);
```

Parameters

Parameter	Description
-----------	-------------

windowInfo	Pointer to a Citrix Window Information object containing window data.
------------	---

Example

```
CtxWI *CWI_7001c = new CtxWI(0x1001c, "Warning !!", 299, 139, 427, 351);
...
CtxDoubleClick(CWI_7001c);
```

CtxKeyDown

Inputs the key stroke specified by the key argument, which corresponds to the VK code. Ensure that *wi* is the foreground window.

Syntax

```
void CtxKeyDown(const CtxWI *wi, int key);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
key	Virtual Key code to type.

Example

```
CtxWI *CWI_2006c = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
CtxKeyDown(CWI_2006c, 107); // '+'
DO_MSLEEP(93);
CtxKeyUp(CWI_2006c, 107); // '+'
```

CtxKeyUp

Inputs the key stroke specified by the key argument, which corresponds to the VK code. Ensure that *wi* is the foreground window.

Syntax

```
void CtxKeyUp(const CtxWI *wi, int key);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
key	Virtual Key code to type.

Example

```
CtxWI *CWI_2006c = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
CtxKeyDown(CWI_2006c, 103); // '7'
DO_MSLEEP(93);
CtxKeyUp(CWI_2006c, 103); // '7'
```

CtxMouseDown

Presses the specified mouse button.

Syntax

```
void CtxMouseDown(const CtxWI* windowInfo, long button, long ModifierKeys, long Xpos, long Ypos);
```

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
button	A mouse button to use for this action. NONE: No mouse button was specified L_BUTTON: The left mouse button R_BUTTON: The right mouse button M_BUTTON: The middle mouse button
ModifierKeys	A keyboard modifier to use for this action. NONE: No keyboard modifier was specified SHIFT: Shift key CONTROL: Control key ALT: Alt key EXTENDED: An extended key
Xpos	Move the mouse to this X coordinate.
Ypos	Move the mouse to this Y coordinate.

Example

```
CtxWI *CWI_2006c = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
CtxMouseDown(CWI_2006c, L_BUTTON, NONE, 274, 316);
DO_MSLEEP(109);
CtxMouseUp(CWI_2006c, L_BUTTON, NONE, 274, 316);
```

CtxMouseMove

Move the mouse to the specified location on the screen.

Syntax

```
void CtxMouseMove(long x, long y);
```

Parameters

Parameter	Description
x	Move the mouse to this X coordinate.
y	Move the mouse to this Y coordinate.

Example

```
CtxMouseMove(274, 316);
```

CtxMouseUp

Releases the specified mouse button.

Syntax

```
void CtxMouseUp(const CtxWI* windowInfo, long button, long ModifierKeys, long Xpos, long Ypos);
```

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
button	A mouse button to use for this action. NONE: No mouse button was specified L_BUTTON: The left mouse button R_BUTTON: The right mouse button M_BUTTON: The middle mouse button
ModifierKeys	A keyboard modifier to use for this action. NONE: No keyboard modifier was specified SHIFT: Shift key CONTROL: Control key ALT: Alt key EXTENDED: An extended key
Xpos	Move the mouse to this X coordinate.
Ypos	Move the mouse to this Y coordinate.

Example

```
CtxWI *CWI_2006c = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
CtxMouseDown(CWI_2006c, L_BUTTON, NONE, 274, 316);
DO_MSLEEP(109);
CtxMouseUp(CWI_2006c, L_BUTTON, NONE, 274, 316);
```

CtxPing

Sends a ping request and wait for the response.

Syntax

```
void CtxPing(const char *identifier);
```

Parameters

Parameter	Description
*identifier	A string to send to the server.

Example

```
CtxPing("7");
```

CtxPoint

See also [Citrix](#)

Moves the mouse to the specified location on the screen.

Syntax

```
void CtxPoint(long X, long Y);
```

Parameters

Parameter	Description
X	X coordinate.
Y	Y coordinate.

Example

```
CtxPoint(509, 422);
```

CtxScreenEventExists

Waits for the specified screen update to occur at the specified coordinates.

Syntax

```
BOOL CtxScreenEventExists(char *EventType, int nmWait, const char *EventInfo);
```

Parameters

Parameter	Description
EventType	Citrix screen event. Valid values are: EVT_STR_CTXSCREENUPDATE: "ScreenUpdate"
nmWait	Amount of time in milliseconds to wait for the event.
*wi	Event string.

Example

```
CtxClick(CWI_2, 188, L_BUTTON, NONE); //1087322563.276
    if(CtxScreenEventExists(EVT_STR_CTXSCREENUPDATE,3000,"0 0 224 3910"))
        BeginBlock();
            RR_printf("Screen Update Found Test1");
        EndBlock();
```

CtxSetApplication

Connects to the Citrix server with the specified application name and application working directory name.

A session is created on the server where only the specified application appears. The working directory parameter is optional. Specify NULL for no working directory.

Syntax

```
void CtxSetApplication (const char *appName, const char *dirName);
```


Parameters

Parameter	Description
*appName	The application to start up after connecting.
*dirName	The working directory for the application.

Examples

```
//Create a session containing only the application called
// application.exe. No working directory is specified.
CtxSetApplication("c:\\stuff\\application.exe", NULL);
```

CtxSetCitrixPort

Sets the port for the Citrix client to use to connect to the server.

Syntax

```
void CtxSetCitrixPort (int port);
```

Parameters

Parameter	Description
port	The port number.

Example

```
CtxSetCitrixPort (1494);
```

CtxSetConnectTimeout

Sets the number of seconds to wait for connections to the server to complete.

Syntax

```
void CtxSetConnectTimeout(int timeout);
```

Parameters

Parameter	Description
timeout	Number of seconds to wait when attempting to connect.

Example

```
//Use a timeout of 1 minute, 30 seconds for connect
CtxSetConnectTimeout(90);
```

CtxSetDisconnectTimeout

Sets the number of seconds to wait for disconnections from the server to complete.

Syntax

```
void CtxSetDisconnectTimeout(int timeout);
```

Parameters

Parameter	Description
timeout	Number of seconds to wait when attempting to disconnect.

Example

```
//Use a timeout of 1 minute, 30 seconds for disconnect
CtxSetDisconnectTimeout(90);
```

CtxSetDomainLoginInfo

Connects to the Citrix server with the specified user name, password, and domain.

Syntax

```
void CtxSetDomainLoginInfo (const char *username, const char *password, const char *domain);
```

Parameters

Parameter	Description
*username	The user name to use when connecting.
*password	The password to use when connecting.
*domain	The domain to use when connecting.

Example

```
const char *CitrixUsername="citrix";
// QALoad "encrypts" the password in the script. The
// login functions also support regular strings.
const char *CitrixPassword    ="~encr~657E06726F697206";
const char *CitrixDomain      ="domain3";
...
CtxSetDomainLoginInfo (CitrixUsername, CitrixPassword, CitrixDomain);
```

CtxSetEnableCounters

Enables or disables custom counters for Citrix client-side statistics.

Syntax

```
void CtxSetEnableCounters (BOOL enable);
```

Parameters

Parameter	Description
enable	TRUE or FALSE

Example

```
CtxSetEnableCounters (TRUE);
```

CtxSetEnableWildcardMatching

Enables or disables wildcard and substring name comparisons for matching Citrix window creation events.

Syntax

```
void CtxSetEnableWildcardMatching (BOOL enable);
```

Parameters

Parameter	Description
enable	TRUE or FALSE

Example

```
CtxSetEnableWildcardMatching (TRUE); //Wildcards are enabled for this script
BEGIN_TRANSACTION();
```

See [CtxSetWindowMatchTitle](#) for another example of wildcards.

CtxSetICAFile

Uses the specified ICA file when connecting to a published application or desktop.

The ICA file can be used to specify a number of configuration options. A Citrix MetaFrame administrator can provide an ICA file for your environment.

Syntax

```
void CtxSetICAFile (const char *filename);
```

Parameters

Parameter	Description
filename	The unqualified name or URL for the ICA file to use.

Example

```
CtxSetICAFile ("published-app.ica");
```

CtxSetLoginInfo

Connects to the Citrix server with the specified user name and password.

Syntax

```
void CtxSetLoginInfo (const char *username, const char *password);
```

Parameters

Parameter	Description
*username	The user name to use when connecting.
*password	The password to use when connecting.

Example

```
const char *CitrixUsername = "citrix";
```

QALoad 5.02

```
// QALoad "encrypts" the password in the script. The
// login functions also support regular strings.
const char *CitrixPassword = "~encr~657E06726F697206";
...
CtxSetLoginInfo (CitrixUsername, CitrixPassword);
```

CtxSetPingTimeout

Sets the number of seconds to wait for a ping to be acknowledged.

Syntax

```
void CtxSetPingTimeout (int timeout);
```

Parameters

Parameter	Description
timeout	The number of seconds to wait for the ping to be acknowledged.

Example

```
CtxSetPingTimeout (30);
```

CtxSetWaitPointTimeout

Sets the number of seconds to wait for a wait point.

Syntax

```
void CtxSetWaitPointTimeout (int timeout);
```

Parameters

Parameter	Description
timeout	Number of seconds to wait for a waitpoint.

Example

```
CtxSetWaitPointTimeout (30);
```

CtxSetWindowMatchTitle

Sets the string to match the names of previously-created windows in the [WaitForWindowCreate](#) method of the Citrix Window Information object.

This name is used for comparison if wildcards have been enabled by the [SetEnableWildcardMatching](#) method call earlier in the script.

Syntax

```
void CtxSetWindowMatchTitle (CtxWI* windowInfo, char* strWindowMatchName);
```

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object that contains window data.

strWindowMatchName	A character string of the name to match with wildcards.
--------------------	---

Example

```
CtxSetWindowMatchTitle(CWI_1, "*Microsoft Word"); //Sets the wildcard match name

CtxWaitForWindowCreate(CWI_1); //With the match name set above, finding
//any current window with a title ending in "Microsoft Word" will match
//and allow this function to return successfully.
```

CtxSetWindowRetries

Sets the retry information for window verification.

If a window does not exist initially, the middleware waits for the number of milliseconds specified before retrying. The verification takes place for the number of times specified.

Syntax

```
void CtxSetWindowRetries(int retries, int waittime);
```

Parameters

Parameter	Description
retries	The number of times to retry verifying the window.
waittime	The number of milliseconds to wait between retries.

Example

```
//Use 3 retries with a 3-second delay
CtxSetWindowRetries(3, 3000);
```

CtxSetWindowTimeout

Set the number of seconds to wait for windows to be activated and destroyed.

Syntax

```
void CtxSetWindowTimeout (int timeout);
```

Parameters

Parameter	Description
timeout	Number of seconds to wait for a window to be created.

Example

```
CtxSetWindowTimeout (30);
```

CtxSetWindowVerification

Enables or disables window verification for actions.

Syntax

```
void CtxSetWindowVerifiatiion (BOOL enable);
```

Parameters

Parameter	Description
enable	TRUE or FALSE

Example

```
CtxSetWindowVerification (TRUE);
```

CtxType

Inputs the specified key strokes. If Window Verification is enabled, ensure that `wi` is the foreground window.

Syntax

```
void CtxType(const CtxWI *wi, char *text);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*text	ASCII text to type into the window.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
CtxType(CWI_40034, "HELLO");
```

CtxTypeChar

See also [Citrix](#)

Sends the specified ASCII character to the Citrix server.

Inputs the key stroke specified by the key argument, which corresponds to the character. Ensure that `wi` is the foreground window. This is equivalent to `KeyDown(key)` and `KeyUp(Key)`.

Syntax

```
void CtxTypeChar(const CtxWI *wi, long vk, int mod);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
vk	ASCII character to type.
mod	A keyboard modifier to use for this action.

Example

```
CtxTypeChar(CWI_3001e, 'F', ALT); //Send ALT-F
```

CtxTypeVK

Sends the VK code that corresponds to a key typed by the user.

Inputs the key stroke specified by the key argument, which corresponds to the VK code. Ensure that wi is the foreground window. This is equivalent to `KeyDown(key)` and `KeyUp(Key)`.

Syntax

```
void CtxTypeVK(const CtxWI *wi, long key, int mod);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
key	Virtual Key code to type.
mod	KEY_MODIFIER

Example

```
CtxTypeVK(CWI_3001e, VK_RIGHT, EXTENDED); //Send the right arrow key
```

CtxWaitForCaptionChange

Waits for the specified window's caption to be changed.

Syntax

```
void CtxWaitForCaptionChange(const CtxWI *wi, const char *newcaption, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*newcaption	String representing new window caption.
nmWait	Amount of time to wait for the event.

Example

```
DO_MSLEEP (234);
// Wait for the title of the window that matches CWI_11 to change to "new title"
CtxWaitForCaptionChange (CWI_11, "new title", 2000);
DO_MSLEEP (125);
```

CtxWaitForScreenUpdate

Waits for the specified screen update to occur at the specified coordinates.

Syntax

```
void CtxWaitForScreenUpdate(long x, long y, long w, long h, long nmWait);
```

Parameters

Parameter	Description
x	X coordinate
y	Y coordinate
w	Width of the screen update
h	Height of the screen update
nmWait	Amount of time in milliseconds to wait for the event

Example

```
CtxWaitForScreenUpdate(154, 154, 253, 261, 500);
```

CtxWaitForWindowActivate

Waits for the specified window to be activated (brought to the foreground).

Syntax

```
void CtxWaitForWindowActivate(const CtxWI *wi, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowActivate(CWI_40034, 500);
```

CtxWaitForWindowCreate

Waits for the specified window to be created.

Syntax

```
void CtxWaitForWindowCreate(const CtxWI *wi, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
```



```
...
CtxWaitForWindowCreate(CWI_40034, 500);
```

CtxWaitForWindowDestroy

Waits for the specified window to be destroyed.

Syntax

```
void CtxWaitForWindowDestroy(const CtxWI *wi, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowDestroy(CWI_40034, 500);
```

CtxWaitForWindowLgIconChange

Waits for the specified window's caption to be changed.

Syntax

```
void CtxWaitForWindowLgIconChange(const CtxWI *wi, const char *hash, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*hash	Unique hash of icon bitmap.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
// Window CWI_13 ("Windows Task Manager") created 1101909450.125
CtxWaitForWindowCreate(CWI_13, 110);
CtxPoint(327, 2); //1101909452.531
CtxWaitForWindowLgIconChange(CWI_13, "c48b65c8b825324a5ff73638118fb8fc", 1218);
```

CtxWaitForWindowMinimize

Waits for the specified window to be minimized.

Syntax

```
void CtxWaitForWindowMinimize(const CtxWI *wi, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowMinimize(CWI_40034, 500);
```

CtxWaitForWindowMove

Waits for the specified window to be moved to the specified coordinates.

Syntax

```
void CtxWaitForWindowMove(const CtxWI *wi, long x, long y, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data
x	X coordinate
y	Y coordinate
nmWait	Amount of time in milliseconds to wait for the event

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowMove(CWI_40034, 132, 297, 2000);
```

CtxWaitForWindowResize

Waits for the specified window to be resized to the specified dimensions.

Syntax

```
void CtxWaitForWindowResize(const CtxWI *wi, long w, long h, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
w	X coordinate.

h	Y coordinate.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowResize(CWI_40034, 100, 200, 500);
```

CtxWaitForWindowSmIconChange

Waits for the specified window's caption to be changed.

Syntax

```
void CtxWaitForWindowSmIconChange(const CtxWI *wi, const char *hash, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*hash	Unique hash of icon bitmap.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
// Window CWI_13 ("Windows Task Manager") created 1101909450.125
CtxWaitForWindowCreate(CWI_13, 110);
CtxPoint(327, 2); //1101909452.531
CtxWaitForWindowSmIconChange(CWI_13, "924f75dd0db6ecb28d3e513053a8038e", 1391);
```

CtxWaitForWindowStyleChange

Waits for the specified window's style to be changed as specified.

Syntax

```
void CtxWaitForWindowStyleChange(const CtxWI *wi, long style, long extendedStyle, long nmWait);
```

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
style	Bit mask that corresponds to the window style.
extendedStyle	Bit mask that corresponds to the extended window style.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
Point (155, 1);
DO_MSLEEP (515);
//Wait for the window that matches CWI_11 to maximize
CtxWaitForWindowStyleChange (CWI_11, 0x14ca0044, 0x50100, 500);
DO_MSLEEP (11047);
```

CtxWindowEventExists

Checks to see if the specified window event has already occurred and, if not, waits for the specified time for the event to occur.

Syntax

```
BOOL CtxWindowEventExists(char *EventType, int nmWait, const CtxWI *wi);
```

Parameters

Parameter	Description
EventType	Citrix window event. Valid values are: EVT_STR_CTXWINDOWCREATE: "WindowCreate" EVT_STR_CTXWINDOWACTIVATE: "WindowActivate" EVT_STR_CTXWINDOWMOVE: "WindowMove" EVT_STR_CTXWINDOWDEACTIVATE: "WindowDeactivate" EVT_STR_CTXWINDOWDESTROY: "WindowDestroy" EVT_STR_CTXWINDOWSIZE: "WindowResize" EVT_STR_CTXWINDOWMINIMIZE: "WindowMinimize" EVT_STR_CTXWINDOWCAPTIONCHANGE: "WindowCaptionChange" EVT_STR_CTXWINDOWSMALLICONCHANGE: "WindowSmallIconChange" EVT_STR_CTXWINDOWLARGEICONCHANGE: "WindowLargeIconChange" EVT_STR_CTXWINDOWSTYLECHANGE: "WindowStyleChange"
nmWait	Amount of time in milliseconds to wait for the event.
*wi	Pointer to a Citrix Window Information object containing window data.

Example

```
// Window CWI_15 ("Open") destroyed 1087837404.827
if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE,3000,CWI_16))
BeginBlock();
    CtxPoint(337, 265); //1087837404.905
    // Window CWI_16 ("11111111 - Microsoft Word") created 1087837404.905
    CtxWaitForWindowCreate(CWI_16, 31);
    // Window CWI_14 ("Document1 - Microsoft Word") destroyed 1087837404.905
    DO_MSLEEP(7547);
    CtxPoint(628, 9); //1087837414.592
    DO_MSLEEP(2141);
    CtxClick(CWI_16, 281, L_BUTTON, NONE); //1087837414.873
    DO_MSLEEP(234);
    // Window CWI_16 ("11111111 - Microsoft Word") destroyed 1087837415.108
```

```

    CtxPoint(113, 93); //1087837418.779
    // Window CWI_17 ("") created 1087837418.779
EndBlock()

```

EndBlock

End of an else block of code.

Syntax

```
void EndBlock();
```

Return Value

None

Parameters

None

Example

```

// Window CWI_5 ("Citrix License Warning Notice") created 1087837373.062
if(CtxWindowEventExists(EVT_STR_CTXWINDOWCREATE, 3000, CWI_5))
BeginBlock();
CtxWaitForWindowCreate(CWI_5, 46);
EndBlock();

```

DB2

DB2 Index

DO_FreeDB2

Releases the memory used by QALoad 's DB2 driver. It should only be called once at the end of a script.

DO_initDB2

Initializes QALoad 's internal DB2 variables. Must be called at the beginning of the script prior to any other calls.

DO_SQLBindFileToCol

SQLBindFileToCol() is used to bind a LOB column in a result set to a file reference or an array of file references. This allows data in that column to be transferred directly into a file when each row is fetched for the statement handle.

DO_SQLBindFileToParam

SQLBindFileToParam() binds a file to a particular parameter in a statement that will be executed.

DO_SQLBindParameter

Used mainly when binding output parameters from stored procedures.

DO_SQLBuildDataLink

Used to build a datalink value.

DO_SQLGetConnectAttr

Gets a characteristic of a connection.

DO_SQLGetDataLinkAttr

DO_SQLGetDataLinkAttr is used with a datalink value that has been retrieved from the database or built using SQLBuildDataLink. The attribute value determines the attribute from the datalink that is returned.

DO_SQLGetLength

DO_SQLGetLength determines the amount of space needed for a string to be sent over. The String values can be of different types.

DO_SQLGetPosition

SQLGetPosition() returns the starting position of one string within a LOB value (the source).

DO_SQLGetSmtAttr

Gets a characteristic of a statement.

DO_SQLGetSubString

Used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call during the current transaction.)

DO_SQLParamData

Used in conjunction with DO_SQLPutData to supply parameter data at statement execution time.

DO_SQLPutData

DO_SQLPutData() is called following a SQLParamData() call returning SQL_NEED_DATA to supply parameter data values. This function can be used to send large parameter values in pieces.

DO_SQLSetConnection

DO_SQLSetConnection allows a user to specify the current active connection. This function will only be used in cases with multiple open connections.

DO_FreeDB2

Releases the memory used by QALoad 's DB2 driver. It should only be called once at the end of a script. This function is the cleanup function for DB2 scripts. There are many stages to cleanup. The first stage in the cleanup is to free all malloced column attributes for each given statement. All of the columns and parameters for each of the statements that have been allocated are freed. Finally each open connection is closed and freed.

DO_FreeDB2 takes a single argument, the PLAYERINFO structure, and should only be called once.

Syntax

```
DO_FreeDB2( sInfo );
```

Parameters

Parameter	Description
sInfo	Structure used by each virtual user.

Example

```
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
DO_SQLDisconnect( C0 );
END_TRANSACTION();
DO_FreeDB2(sInfo);
REPORT(SUCCESS);
EXIT();
```

DO_initDB2

Initializes QALoad 's internal DB2 variables. Must be called at the beginning of the script prior to any other calls.

Syntax

```
DO_initDB2(bHandleLOBs, sInfo );
```

Parameters

Parameter	Description
bHandleLOBs	Determines how QALoad will deal with LOBs. In calls to SQLBindFileToParam, the script uses a default file as the LOB being sent into the database. This file is of a known length so it can be used for each different type of LOB handled by DB2.
sInfo	This needs to be passed in order to properly initialize the Thread Local Storage.

Example

```
SET_ABORT_FUNCTION( abort_function );
DO_initDB2( FALSE, sInfo );
SYNCHRONIZE();
BEGIN_TRANSACTION();
DO_SLEEP(2);
DO_SQLConnect( C0, "DB2RGR", "db2sa", "db2sa" );
```

DO_SQLBindFileToCol

SQLBindFileToCol() is used to bind a LOB column in a result set to a file reference or an array of file references. This allows data in that column to be transferred directly into a file when each row is fetched for the statement handle.

The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before fetching each row, the application must make sure that these variables contain the name of a file, the length of the file name, and a file option (new/overwrite/append). These values can be changed between each fetch.

Syntax

```
DO_SQLBindFileToCol( nStatementIndex, nColumnNumber,
                    sFileName, nFileNameLength,
                    nFileOptions, nMaxFileNameLength );
```

Parameters

Parameter	Description
nStatementIndex	The index for the statement handle structure that this function call is part of.
nColumnNumber	Number identifying the column. Columns are numbered sequentially, from left to right, starting at 1.
sFileName	The name of the file that will receive the LOB from the statement execution.

nFileNameLength	The length of the file. The maximum value of the file name length is 255.
nFileOptions	The file options determine the behavior of the file: whether this should create the file, append to the file, or whether it should overwrite the file. These are the behaviors that the SQLBindFileToCol can exhibit. SQL_FILE_CREATE Create a new file. If a file by this name already exists, SQL_ERROR will be returned. SQL_FILE_OVERWRITE If the file already exists, overwrite it. Otherwise, create a new file. SQL_FILE_APPEND If the file already exists, append the data to it. Otherwise, create a new file. Only one option can be chosen per file, there is no default.
nMaxFileNameLength	This specifies the length of the FileName buffer.

Example

```
/* Bind blob column to a file */
rc = SQLBindFileToCol(hstmt, 1, FName, &FNLength, &FOption, 13, NULL, &FNInd);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );
```

DO_SQLBindFileToParam

SQLBindFileToParam() binds a file to a particular parameter in a statement that will be executed.

SQLBindFileToParam() is used to associate (bind) a parameter marker in an SQL statement to a file reference or an array of file references. This enables data from the file to be transferred directly into a LOB column when that statement is executed. The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before calling SQLExecute() or SQLExecDirect(), the application must make sure that this information is available in the deferred input buffers. These values can be changed between SQLExecute() calls.

Syntax

```
DO_SQLBindFileToParam( nStatementIndex, ParaNumber, nSQLType,
                      sFileName, nFileNameLength,
                      nFileOptions, nMaxFileNameLength, pIndicator );
```

Parameters

Parameter	Description
nStatement	Index Statement Handle.
ParamNumber	Parameter marker number. Parameters are numbered sequentially, from left to right, starting at 1.
nSQLType	SQL Data Type of the column. The data type must be: SQL_BLOB, SQL_CLOB, SQL_DBCLOB.
sFileName	Pointer to the location that will contain the file name or an array of file names when the statement (StatementHandle) is executed. This is either the complete path name of the file or a relative file name. If a relative file name is provided, it is appended to the current path of the client process.

	This argument cannot be NULL.
nFileNameLength	Pointer to the location that will contain the length of the file name (or an array of lengths) at the time of the next <code>SQLExecute()</code> or <code>SQLExecDirect()</code> using the <code>StatementHandle</code> . If this pointer is NULL, then a length of <code>SQL_NTS</code> is assumed. The maximum value of the file name length is 255.
nFileOptions	Pointer to the location that will contain the file option (or an array of file options) to be used when reading the file. The location will be accessed when the statement (<code>StatementHandle</code>) is executed. Only one option is supported (and it must be specified): <code>SQL_FILE_READ</code> . A regular file that can be opened, read and closed. (The length is computed when the file is opened.) This pointer cannot be NULL.
nMaxFileNameLength	This specifies the length of the <code>FileName</code> buffer. If the application calls <code>SQLParamOptions()</code> to specify multiple values for each parameter, this is the length of each element in the <code>FileName</code> array.
pIndicator	Pointer to the location that contains an indicator value (or array of values), which is set to <code>SQL_NULL_DATA</code> if the data value of the parameter is to be null. It must be set to 0 (or the pointer can be set to null) when the data value is not null.

Example

```
/* Bind the Blob file to the parameter */
rc = SQLBindFileToParam (hstmt, 3, SQL_BLOB, FName,
                        &FNlength, &FOption, 255, &FInd) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

DO_SQLBindParameter

Used mainly when binding output parameters from stored procedures.

Bind parameters are identified in an SQL statement with the question mark (?) character. Each ? character is a separate bind parameter, with the first bind parameter starting at 1.

Return type

```
SQL_PARAM_INPUT
SQL_PARAM_INPUT_OUTPUT
SQL_PARAM_OUTPUT
```

C data type

The following constants should be used when specifying the C data type:

```
SQL_C_BINARY
SQL_C_LONG
SQL_C_BIT
SQL_C_DEFAULT
SQL_C_BLOB_LOCATOR
SQL_C_SBIGINT
SQL_C_CHAR
SQL_C_SHORT
SQL_C_CLOB_LOCATOR
SQL_C_TYPE_DATE
SQL_C_DBCHAR
```

QALoad 5.02

SQL_C_TYPE_TIME
SQL_C_DBCLOB_LOCATOR
SQL_C_TYPE_TIMESTAMP
SQL_C_DOUBLE
SQL_C_TINYINT
SQL_C_FLOAT
SQL_C_UBIGINT

SQL data type

SQL_BIGINT
SQL_BINARY
SQL_BLOB
SQL_BLOB_LOCATOR
SQL_CHAR
SQL_CLOB
SQL_CLOB_LOCATOR
SQL_DBCLOB
SQL_DBCLOB_LOCATOR
SQL_DECIMAL
SQL_DOUBLE
SQL_FLOAT
SQL_GRAPHIC
SQL_INTEGER
SQL_LONGVARIABLE
SQL_LONGVARIABLE
SQL_LONGVARIABLE
SQL_NUMERIC
SQL_REAL
SQL_SMALLINT
SQL_TYPE_DATE
SQL_TYPE_TIME
SQL_TYPE_TIMESTAMP
SQL_VARIABLE
SQL_VARIABLE
SQL_VARIABLE

Dates and times are input in the following formats:

SQL_DATE: YYYY:MM:DD (for example: 1996:10:25)
SQL_TIME: HH:MM:SS (for example: 17:28:01)
SQL_TIMESTAMP: YYYY:MM:DD:HH:MM:SS

Syntax

```
DO_SQLBindParameter( CommandIndex, nParamNum,  
                    ParamType, CDataType, DataType,  
                    ColumnDefinition, Scale,  
                    InputString, cbValueMax );
```

Parameters

Parameter	Description
CommandIndex	Index into the table of ODBC command handles.
nParamNum	Bind parameter number, first parameter is 1.
ParamType	Defines the direction of the parameter input, output, or input/output of the bind call.
CDataType	C data type.
DataType	SQL data type.

ColumnDefinition	Precision of the column. (The same as using the native ODBC call for the given C or SQL type.)
Scale	Scale of the column. (The same as using the native ODBC call for the given C or SQL type.)
InputString	A character string that defines the input data to the bind call.
cbValueMax	The length of the output variable being passed.


Example

```
#define MY_SIZE 16

char* sRow1 = NULL;
char* sRow2 = NULL;
char* sRow3 = NULL;
char* sRow4 = NULL;

DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloaddb2", "sa", "" );
DO_SQLAllocStmt( C0, S0 );
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT,
SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 2, SQL_PARAM_OUTPUT,
SQL_C_ULONG, SQL_Integer, 0, 0, 4 );
DO_SQLBindParameter( S0, 3, SQL_PARAM_OUTPUT,
SQL_C_ULONG, SQL_Integer, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 4, SQL_PARAM_OUTPUT,
SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 5, SQL_PARAM_OUTPUT,
SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
strcpy(sql_statement, "{call setup_rows (?,?,?,?) }" );
DO_SQLPrepare( S0, sql_statement );
DO_LoadMem( S0, 1, "17", 4 );
DO_LoadMem( S0, 2, "1234", 4 );
DO_LoadMem( S0, 3, "1235", 4 );
DO_LoadMem( S0, 4, "1236", 4 );
DO_LoadMem( S0, 5, "1237", 4 );
DO_SQLExecute( S0 );

sRow1 = DO_SQLRetrieveParamValue( S0, 2 );
sRow2 = DO_SQLRetrieveParamValue( S0, 3 );
sRow3 = DO_SQLRetrieveParamValue( S0, 4 );
sRow4 = DO_SQLRetrieveParamValue( S0, 5 );
```

 **Note:** Make sure to free each of the char* local variables used for parameter retrieval at the end of each transaction loop.

DO_SQLBuildDataLink

Used to build a datalink value.

DO_SQLBuildDataLink is necessary for load testing for applications that use datalinks. A datalink is a logical reference from the database to a file stored outside of the interface. The maximum length of the string including the null terminating character will be the bufferlength of bytes.

Syntax

```
DO_SQLBuildDataLink( nStatementIndex, sLinkType,
                    nLinkTypeLen, sDataLinkLocation,
```

```
nDataLinkLocationLength, sComment,
nCommentLength, nDataLinkValueLength );
```

Parameters

Parameter	Description
nStatementIndex	Statement Handle.
sLinkType	This is always set to SQL_DATALINK_URL.
nLinkTypeLen	Should always be set to the length of the sLinkType argument.
sDataLinkLocation	The complete URL value to be assigned.
nDataLinkLocationLength	Set to the length of the sDataLinkLocation.
sComment	A Comment to be assigned to the datalink.
nCommentLength	The length of the sComment argument.
nDataLinkValueLength	The length of the buffer assigned returned value.

DO_SQLGetConnectAttr

Gets a characteristic of a connection.

Syntax

```
DO_SQLGetConnectAttr( nConnectionIndex, nAttribute, nBuffer Length );
```

Parameters

Parameter	Description
nConnectionIndex	Index of a connection handle.
nAttribute	For example: SQL_ATTR_AUTOCOMMIT is an attribute with set values SQL_TRUE or SQL_FALSE. Other connection attributes have different values. Note that QALoad does not allow SQL_ATTR_ENABLE_ASYNC to be true for DB2. No asynchronous transactions will be handled.
nBufferLength	Length of the return value.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_DBC, 0, C0 );
DO_SQLConnect( C0, "DB2MAST", "db2sa", "db2sa" );
DO_SLEEP( 1 );
DO_SQLSetConnectAttr( C0, SQL_ATTR_AUTOCOMMIT, "SQL_AUTOCOMMIT_ON", SQL_IS_INTEGER );
DO_SQLGetConnectAttr( C0, 102, 0 );
```

DO_SQLGetDataLinkAttr

DO_SQLGetDataLinkAttr is used with a datalink value that has been retrieved from the database or built using SQLBuildDataLink. The attribute value determines the attribute from the datalink that is returned. This function should not be necessary for a load test.

Syntax

```
DO_SQLGetDataLinkAttr ( nStatementIndex, sAttributeType, sDataLinkName, nDataLinkNameLength,
nBufferLength );
```

Parameters

Parameter	Description
nStatementIndex	Corresponds to the statement index.
sAttributeType	The following are the different Attribute types that are used with this function: SQL_ATTR_DATALINK_COMMENT SQL_ATTR_DATALINK_LINKTYPE SQL_ATTR_DATALINK_URLCOMPLETE SQL_ATTR_DATALINK_URLPATH SQL_ATTR_DATALINK_URLPATHONLY SQL_ATTR_DATALINK_URLSCHEME SQL_ATTR_DATALINK_URLSERVER
sDataLinkName	The name of the datalink.
nDataLinkNameLength	The length of the datalink name.
nBufferLength	The length of the data being returned as the datalink value.

DO_SQLGetLength

DO_SQLGetLength determines the amount of space needed for a string to be sent over. The String values can be of different types.

DO_SQLGetLength uses 3 parameters as inputs: the StatementIndex, the LOB Type, and a LOB Locator.

Syntax

```
DO_SQLGetLength ( nStatementIndex, LOBType, nLocator);
```

Parameters

Parameter	Description
nStatementIndex	Statement Index. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
LOBType	The C type of the source LOB locator. This may be: SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR SQL_C_DBCLOB_LOCATOR
nLocator	Must be set to the LOB locator value.

Example

```
DO_SQLGetLength ( S3, SQL_C_CLOB_LOCATOR, 27288774);
DO_SQLGetPosition( S3, SQL_C_CLOB_LOCATOR, 27288774, 1202, "artist", 6, 1 );
DO_SQLGetSubString( S3, SQL_C_CLOB_LOCATOR, 27288774, 398, 100, SQL_C_CHAR, 110 );
```

DO_SQLGetPosition

SQLGetPosition() returns the starting position of one string within a LOB value (the source).

The source value must be a LOB locator. The search string can be a LOB locator or a literal string. The source and search LOB locators can be any that have been returned from the database from a fetch or a SQLGetSubString() call during the current transaction.

Syntax

```
DO_SQLGetPosition( StatementIndex, LOBType, LOBLocator, SearchLocator, SearchLiteral,
SearchLiteralLength, nFromPosition 1 );
```

Parameters

Parameter	Description
nStatementIndex	Statement Index. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
LOBType	The type of LOB that is being searched in the call. The C type of the source LOB locator. This may be: SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR SQL_C_DBCLOB_LOCATOR
LOBLocator	A DB2-generated number that gives a location within a database where this LOB is stored.
nSearchLocator	If the SearchLiteral pointer is NULL and the SearchLiteralLength is set to 0, then SearchLocator must be set to the LOB locator associated with the search string otherwise, this argument is ignored.
nSearchLiteral	This argument points to the area of storage that contains the search string literal. If SearchLiteralLength is 0, this pointer must be NULL.
nSearchLiteralLength	The length of the string in SearchLiteral(in bytes). If this argument value is 0, then the argument SearchLocator is meaningful.
nFromPosition	For BLOBs and CLOBs, this is the position of the first byte within the source string at which the search is to start, to be returned by the function. For DBCLOBs, this is the first character. The start byte or character is numbered 1.

Example

```
DO_SQLGetLength( S3, SQL_C_CLOB_LOCATOR, 27288774 );
DO_SQLGetPosition( S3, SQL_C_CLOB_LOCATOR, 27288774, 1202, "Elvis", 5, 1 );
DO_SQLGetSubString( S3, SQL_C_CLOB_LOCATOR, 27288774, 398, 100, SQL_C_CHAR, 110 );
```

DO_SQLGetStmtAttr

Gets a characteristic of a statement.

Syntax

```
DO_SQLGetStmtAttr( nStmtIndex, nAttribute, nBufferLength );
```

Parameters

Parameter	Description
nStmtIndex	Index into the table of DB2 statement handles.
nAttribute	For example: SQL_ATTR_APP_ROW_DESC. Note that even at the statement level, QALoad does not permit asynchronous transactions.
nBufferLength	Length of the return value.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLBindCol( S0, 1, SQL_C_Long, 60, 60 );
strcpy( sql_statement, /*>> 1 <<*/ "SELECT keyval FROM testdb.test_table WHERE keyval = {
01} " );
DO_substr( sql_statement, 1, "30" );
DO_SQLExecDirect( S0, sql_statement );
DO_SQLGetStmtAttr( S0, 6, 0 );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLGetSubString

Used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call during the current transaction.)

Syntax

```
DO_SQLGetSubString( nStatementIndex, nLocatorCType, Locator, nFromPosition, nForLength,
nTargetType, nBufferLength )
```

Parameters

Parameter	Description
nStatementIndex	Statement Index. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
sLocatorCType	The C type of the source LOB locator. This may be: SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR SQL_C_DBCLOB_LOCATOR.
Locator	Must be set to the source LOB locator value.
nFromPosition	For BLOBs and CLOBs, this is the position of the first byte within the source string at which the search is to start. To be returned by the function. For DBCLOBs, this is the first character. The start byte or character is numbered 1.
nForLength	The length of the string to be returned by the functions. For BLOBs and CLOBs, this is the length in bytes. For DCLOBs, this is the length in characters. If FromPosition is less than the length of the source string but FromPosition + ForLength - 1 extends beyond the end of the source string, the result is padded on the right with the necessary number of characters.

nTargetType	The C data type of the rgbValue. The target must always be either a LOB indicator C buffer type: SQL_C_CLOB_LOCATOR SQL_C_BLOB_LOCATOR SQL_C_DBCLOB_LOCATOR or a C string variable: SQL_C_Binary SQL_C_Char SQL_C_DBChar.
nBufferLength	A long integer value; the length of the buffer for the data that is being returned.

Example

```
DO_SQLExecute( S2 );
DO_SQLGetLength( S3, SQL_C_CLOB_LOCATOR, 27288774 );
DO_SQLGetPosition( S3, SQL_C_CLOB_LOCATOR, 27288774, 1202, "artist", 6, 1 );
DO_SQLGetSubString( S3, SQL_C_CLOB_LOCATOR, 27288774, 398, 100, SQL_C_CHAR, 110 );
DO_SQLTransact( C0, SQL_COMMIT );
```

DO_SQLParamData

Used in conjunction with DO_SQLPutData to supply parameter data at statement execution time.

Syntax

```
int SQLParamData( int nStatementIndex );
```

Parameters

Parameter	Description
nStmtIndex	Index into the table of DB2 statement handles.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
strcpy(sql_statement, "INSERT INTO dbo.TEST_TABLE (keyval, test_number, longvarchar_col)
VALUES (?, ?, ?)");
DO_SQLPrepare( S0, sql_statement );
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 16, 0, 0,
SQL_DATA_AT_EXEC);
DO_LoadMem( S0, 1, "26293", 0 );
DO_LoadMem( S0, 2, "9", 0 );
DO_SQLExecute( S0 );
DO_SQLParamData( S0 );
DO_SQLPutData( S0, "AAA", -3 );
DO_SQLParamData( S0 );
DO_SQLEndTran( SQL_HANDLE_DBC, C0, SQL_COMMIT );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLSetConnection

DO_SQLSetConnection allows a user to specify the current active connection. This function will only be used in cases with multiple open connections.

Syntax

```
SQLSetConnection( nConnectionIndex );
```

Parameters

Parameter	Description
nConnectionIndex	Points to a structure that ODBC/DB2 uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "DB2RGR", "db2Admin", "db2Admin" );
...
...
...
DO_SQLAllocConnect( C1 );
DO_SQLConnect( C1, "PUB", "db2Admin", "db2Admin" );
...
...
...
DO_SQLSetConnect( C0 );
```

ODBC

ODBC Index

DO_FreeODBC

Releases the memory used by QALoad's ODBC/DB2 driver. It should only be called once at the end of a script.

DO_initODBC

Initializes QALoad's internal ODBC variables. Must be called at the beginning of the script prior to any other calls.

DO_SQLBindParameter

Used to describe a memory location between the application and the database. This memory location is used to exchange data between the application and the database.

DO_FreeODBC

Releases the memory used by QALoad's ODBC/DB2 driver. It should only be called once at the end of a script.

Syntax

```
DO_FreeODBC( sInfo );
```

Parameters

Parameter	Description
sInfo	Structure used by each virtual user.

Example

```
END_TRANSACTION();
DO_FreeODBC( sInfo );
REPORT( SUCCESS );
EXIT();
```

DO_initODBC

Initializes QALoad 's internal ODBC variables. Must be called at the beginning of the script before any other calls.

Syntax

```
DO_initODBC( nVersion, sInfo );
```

Parameters

Parameter	Description
nVersion	This argument deals with the version of ODBC. ODBC uses different functions to allocate and free different structures to handle different properties of connections, statements, and the environment. In order to handle the behavior properly, QALoad detects and passes the version number into the script.
sInfo	This needs to be passed in order to properly initialize the Thread Local Storage.

Example

```
SET_ABORT_FUNCTION( abort_function );
DEFINE_TRANS_TYPE( "wilson.c" );
// Checkpoints have been included by the convert process
DefaultCheckpointsOn();
DO_initODBC( 3, sInfo );
```

DO_SQLBindParameter

Used to describe a memory location between the application and the database. This memory location is used to exchange data between the application and the database.

Bind parameters are identified in a SQL statement with the question mark (?) character. Each ? character is a separate bind parameter, with the first bind parameter starting at 1.

Syntax

```
DO_SQLBindParameter( int CommandIndex, unsigned short nParamNum, short ParamType, short CDataType, short DataType, long ColumnDefinition, short Scale, SDWORD InputString, int cbValueMax );
```

Parameters

Parameter	Description
CommandIndex	Index into the table of ODBC command handles.
nParamNum	Bind parameter number. The first parameter is 1.
ParamType	Defines the direction of the parameter input, output, or input/output of the bind call. Valid values are: SQL_PARAM_INPUT: Parameter is input only

	<p>SQL_PARAM_INPUT_OUTPUT: Parameter is input and output SQL_PARAM_OUTPUT: Parameter is output only</p>
CDataType	<p>C data type. Valid values are:</p> <p>SQL_C_BIT: Bit SQL_C_LONG: Long SQL_C_UTINYINT: Unsigned tiny integer SQL_C_CHAR: Char SQL_C_STINYINT: Signed tiny integer SQL_C_BINARY: Binary SQL_C_TINYINT: Tiny integer SQL_C_FLOAT: Float SQL_C_SSHORT: Signed short SQL_C_DOUBLE: Double SQL_C_USHORT: Unsigned short SQL_C_DATE: Date in YYYY:MM:DD format (for example: 1996:10:25) SQL_C_SHORT: Short SQL_C_TIME: Time in HH:MM:SS format (for example: 17:28:01) SQL_C_SLONG: Signed long SQL_C_TIMESTAMP: Timestamp in YYYY:MM:DD:HH:MM:SS format SQL_C_ULONG: Unsigned long SQL_C_NUMERIC: Numeric</p>
DataType	<p>SQL data type. Valid values are:</p> <p>SQL_CHAR: Char SQL_NUMERIC: Numeric SQL_DECIMAL: Decimal SQL_INTEGER: Integer SQL_SMALLINT: Small integer SQL_FLOAT: Float SQL_REAL: Real SQL_DOUBLE: Double SQL_VARCHAR: Varchar SQL_TIME: Time in HH:MM:SS format (for example: 17:28:01) SQL_LONGVARCHAR: Long varchar SQL_BINARY: Binary SQL_VARBINARY: Variable binary SQL_LONGVARBINARY: Long variable binary SQL_BIGINT: Big integer SQL_TINYINT: Tiny integer SQL_BIT: Bit</p>
ColumnDefinition	<p>Precision of the column. (The same as using the native ODBC call for the given C or SQL type.)</p>
Scale	<p>Scale of the column. (The same as using the native ODBC call for the given C or SQL type.)</p>
InputString	<p>A string that defines the input data to the bind call.</p>
cbValueMax	<p>The length of the output variable being passed.</p>

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
strcpy(sql_statement, "INSERT INTO dbo.TEST_TABLE (keyval, test_number, longvarchar_col)
```

QALoad 5.02

```
VALUES (?, ?, ?)");
DO_SQLPrepare( S0, sql_statement );
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 16, 0, 0,
SQL_DATA_AT_EXEC);
DO_LoadMem( S0, 1, "26293", 0 );
DO_LoadMem( S0, 2, "9", 0 );
DO_SQLExecute( S0 );
DO_SQLParamData( S0 );
DO_SQLPutData( S0, "AAA", -3 );
DO_SQLParamData( S0 );
DO_SQLEndTran( SQL_HANDLE_DBC, C0, SQL_COMMIT );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

ODBC/DB2

ODBC/DB2 Index

DO_LoadMem

Fills the memory location described in a corresponding DO_SQLBindParameter call. The data, sData, is always represented as a string. DO_LoadMem enables sending multiple pieces of data into the same bind call by loading memory that was added with a DO_SQLBindParameter call.

DO_SQLAllocConnect

The connection handle must be allocated before the actual connection can take place. It is important that each DO_SQLAllocConnect call is matched up with a similar DO_SQLFreeConnect, either inside of the transaction loop or outside of the transaction loop.

DO_SQLAllocHandle

Allocates handles. Replaces DO_SQLAllocStmt.

DO_SQLAllocStmt

Allocates a statement handle and assigns it to a previously open connection.

DO_SQLBindCol

Binds application buffers to a specific column of a statement. The columns are identified by number in the result set.

DO_SQLCancel

Cancels the processing of the present SQL statement.

DO_SQLCloseCursor

Closes a cursor associated with a handle and discards the results.

DO_SQLColAttribute

Returns descriptor information for a column in a result set.

DO_SQLColumns

Retrieves the column information of the selected tables.

DO_SQLConnect

Performs a connection to the database.

DO_SQLCopyDesc

If the values of the SourceDescHandle and TargetDescHandle parameters are associated with the same driver, the driver copies all descriptor fields. This is true even if the drivers are on different connections or environments. If the values of the parameters are not associated with the same driver, only ODBC-defined fields are copied.

[DO_SQLDescribeCol](#)

Returns descriptor information to the statement handle.

[DO_SQLDisconnect](#)

Closes the connection from the application to the database server.

[DO_SQLDriverConnect](#)

Connects the application to the database.

[DO_SQLEndTran](#)

Provides the mechanism for all open transactions or all open transactions on a particular connection to be resolved.

[DO_SQLExecDirect](#)

Prepares and executes a SQL statement.

[DO_SQLExecute](#)

Executes a prepared command using the current values of the parameter marker variables, if any parameter markers exist in the command.

[DO_SQLFetch](#)

Retrieves a single row of data.

[DO_SQLFreeConnect](#)

Performs the cleanup of connection handles for ODBC/DB2 within a QALoad script.

[DO_SQLFreeHandle](#)

In ODBC, DO_SQLFreeHandle handles statement and descriptor cleanup. In DB2, DO_SQLFreeHandle handles the additional cleanup of connection handles. Each occurrence of DO_SQLFreeHandle must have a corresponding DO_SQLAllocHandle, either both within the transaction loop or both outside of the transaction loop.

[DO_SQLFreeStmt](#)

Stops processing associated with a specific command_index and:

[DO_SQLGetCursorName](#)

Use on an open ODBC/DB2 statement to return a char * containing the cursor active on a particular statement.

[DO_SQLGetData](#)

Retrieves data for a single column in the form of a string.

[DO_SQLGetDescField](#)

Returns the value of a field of a descriptor record.

[DO_SQLGetDescRec](#)

Returns the settings or values from fields of a descriptor record set by DO_SQLSetDescRec, including name, data type, and column or parameter data storage. Does not retrieve values for header fields.

[DO_SQLGetEnvAttr](#)

Gets a characteristic of an environment.

[DO_SQLGetTypeInfo](#)

Returns information about data types supported by the data source.

[DO_SQLNumResultCols](#)

Determines the number of columns being returned in a result set.

[DO_SQLPrepare](#)

Prepares an SQL statement and associates the results with the `command_index`. The command is not executed until the `DO_SQLExecute` command is called.

[DO_SQLRetrieveParamValue](#)

Retrieves a value of a `SQL_PARAM_INPUT_OUTPUT` or `SQL_PARAM_OUTPUT` parameter, following the execution of the corresponding SQL statement.

[DO_SQLRowCount](#)

Returns an integer indicating the number of rows affected by the last SQL statement associated with the specified `command_index`.

[DO_SQLSetConnectAttr](#)

Sets a characteristic of the connection.

[DO_SQLSetConnectOption](#)

Sets options on the connection handle.

[DO_SQLSetCursorName](#)

Associates a cursor name with an active `command_index`.

[DO_SQLSetDescField](#)

Sets a descriptor field. A call to `DO_SQLSetDescField` can set a field of any descriptor type that can be set.

[DO_SQLSetDescRec](#)

Sets multiple descriptor fields with a single call.

[DO_SQLSetEnvAttr](#)

Sets different aspects of the ODBC environment.

[DO_SQLSetPos](#)

Sets cursor locking and direction properties.

[DO_SQLSetStmtAttr](#)

Sets statement attributes and, as a result, sets descriptor fields.

[DO_SQLSetStmtOption](#)

Sets the boundaries of a specific statement handle.

[DO_SQLSpecialColumns](#)

Retrieves information about columns within a specified table. `DO_SQLSpecialColumns` retrieves the following information:

[DO_SQLStatistics](#)

Retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.

DO_SQLTables

Returns the list of table names stored in a specific data source. The driver returns the information as a result set.

DO_SQLTransact

Requests a commit or rollback operation for all update, insert, and delete transactions in progress on all command indexes associated with a connection. Can also request that a commit or rollback operation be performed for all connections by specifying a connection index of -1.

DO_substr

Finds a value within a string.

GetBindColumnData

Retrieves data from one of the rows that are returned by DO_SQLFetch calls, after a combination of DO_SQLSetStmtAttr and DO_SQLBindCol calls.

Using descriptors

Descriptors are new to ODBC with release ODBC 3.x. They are also present in DB2. They offer a way of tracking column metadata. Descriptors can be used for a number of different purposes, and can be shared by different statements. In most cases, an application doesn't require access to descriptors; however, in some cases accessing descriptors can simplify a number of operations.

There are four types of descriptors:

- ! Application Parameter Descriptor (APD)
Contains either the input parameters set up by the application or the output columns following the execution of a CALL statement within SQL.
- ! Application Row Descriptor (ARD)
Contains the row data as the row is presented to the application.
- ! Implementation Row Descriptor (IRD)
Contains the row as it comes from the database.
- ! Implementation Parameter Descriptor (IPD)
Contains the parameter elements after conversion heading to the database.

For more information on ODBC descriptors, refer to your ODBC 3.0 Programmer's Reference Volume and SDK Guide.

Handling connection descriptors

It is important to note that QALoad processes descriptor handles in the same way it processes statement handles. Each connection handle is associated with a unique descriptor handle. Each time a descriptor is allocated during conversion, QALoad associates descriptors with connections the same way that ODBC and DB2 do.

Connection handle allocation

DB2 now uses DO_SQLAllocHandle, with a SQL_HANDLE_DBC type, or DO_SQLAllocConnect to allocate the connection handle. This allows you to set the connection attributes and options before the actual connection is made. (Previously, connection handle allocation was done with SQLConnect and SQLDriverConnect.)

DO_LoadMem

Fills the memory location described in a corresponding DO_SQLBindParameter call.

The data, sData, is always represented as a string. DO_LoadMem enables sending multiple pieces of data into the same bind call by loading memory that was added with a DO_SQLBindParameter call.

Syntax

```
DO_LoadMem( nStmtIndex, nParamNum, sData, nBufLen );
```

Parameters

Parameter	Description
nStmtIndex	Index into the table of statement handles.
nParamNum	Number of the parameter. This matches the value in DO_SQLBindCol.
sData	String representation of the data.
nBufLength	Length of the string.

Example

```
DO_SQLPrepare( S0, sql_statement );
DO_SQLBindParameter( S0, 1, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 2, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 3, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 4, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 5, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_LoadMem( S0, 1, "22", 4 );
DO_LoadMem( S0, 2, "0", 4 );
DO_LoadMem( S0, 3, "0", 4 );
DO_LoadMem( S0, 4, "0", 4 );
DO_LoadMem( S0, 5, "0", 4 );
DO_SQLExecute( S0 );
```

DO_SQLAllocConnect

Allocates connection handle.

The connection handle must be allocated before the actual connection can take place. It is important that each DO_SQLAllocConnect call is matched up with a similar DO_SQLFreeConnect, either inside of the transaction loop or outside of the transaction loop.

Use DO_SQLAllocHandle in place of DO_SQLAllocConnect if using DB2 or ODBC version 3 or higher.

Syntax

```
DO_SQLAllocConnect( HDBCIndex );
```

Parameters

Parameter	Description
HDBCIndex	Points to a structure that ODBC/DB2 uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "DB2RGR", "db2sa", "db2sa" );
DO_SQLDisconnect( C0 );
DO_SQLFreeConnect( C0 );
```

DO_SQLAllocHandle

Allocates handles. Replaces DO_SQLAllocStmt.

Previous versions of ODBC used different statements to allocate different structures. ODBC 3.x uses a single handle allocation function instead, which is represented by DO_SQLAllocHandle in QALoad . DB2 supports both methods.

Syntax

```
DO_SQLAllocHandle ( handleType , IncomingIndex, OutgoingIndex )
```

Parameters

Parameter	Description
handleType	The type of handle to be allocated. Each handle takes different arguments: SQL_HANDLE_ENV, SQL_HANDLE_DBC (connection handle); SQL_HANDLE_STMT (statement handle); and SQL_HANDLE_DESC (descriptor handle).
IncomingIndex	The structure that the outgoing handle will belong to. An environment handle is the incoming handle for a connection. A connection handle is the incoming handle for statements and for descriptors. When allocating the environment, pass in the following value for the incoming handle argument: SQL_NULL_HANDLE.
OutgoingIndex	Points to the location of the structure that will store information for each of the different structures that ODBC uses.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLBindCol( S0, 1, SQL_C_LONG, 4, 4 );
strcpy(sql_statement, /* >> 2 << */ "select MAX(keyval) from test_table");
DO_SQLExecDirect( S0, sql_statement );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLAllocStmt

Allocates a statement handle and assigns it to a previously open connection.

Syntax

```
DO_SQLAllocStmt( nConnectionIndex, nStatementIndex );
```

Parameters

Parameter	Description
nConnectionIndex	Index into the table of ODBC connection handles.

nStatementIndex	Index into the table of ODBC statement handles.
-----------------	---

Example

```
DO_SQLSetConnectOption( C0, SQL_ACCESS_MODE, 0 );
DO_SQLAllocStmt( C0, S0 );
DO_SQLSetStmtOption( S0, SQL_QUERY_TIMEOUT, 60 );
```

DO_SQLBindCol

Binds application buffers to a specific column of a statement. The columns are identified by number in the result set.

Syntax

```
DO_SQLBindCol( int StatementIndex, int ColumnNum, int CDataType, long BufferLength, long
pBufferLength );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of ODBC statement handles.
ColumnNum	Number of the result set column to bind.
CDataType	The C data type being returned to the application. SQL_C_BIT: Bit SQL_C_UTINYINT: Unsigned tiny integer SQL_C_STINYINT: Signed tiny integer SQL_C_TINYINT: Tiny integer SQL_C_SSHORT: Signed short SQL_C_USHORT: Unsigned short SQL_C_SLONG: Signed long SQL_C_ULONG: Unsigned long SQL_C_LONG: Long SQL_C_CHAR: Char SQL_C_BINARY: Binary SQL_C_FLOAT: Float SQL_C_DOUBLE: Double SQL_C_NUMERIC: Numeric SQL_C_DATE: Date in YYYY:MM:DD format (for example: 1996:10:25) SQL_C_TIME: Time in HH:MM:SS (for example: 17:28:01) SQL_C_TIMESTAMP: Timestamp in YYYY:MM:DD:HH:MM:SS format
BufferLength	The length of the buffer for the data that is being returned.
PBufferLength	The length/indicator buffer to bind to the column.

Example

```
BEGIN_TRANSACTION();

DO_SQLAllocHandle( SQL_HANDLE_DBC, 0, C0 );
DO_SQLConnect( C0, "FHLOADDB2", "sa", "" );
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLSetStmtAttr( S0, SQL_ATTR_ROW_ARRAY_SIZE, 5, SQL_IS_INTEGER ); // Changed from 5 to 0
DO_SQLSetStmtAttr( S0, SQL_ATTR_ROWS_FETCHED_PTR, 0, SQL_IS_POINTER );
DO_SQLSetStmtAttr( S0, SQL_ATTR_ROW_STATUS_PTR, 0, SQL_IS_POINTER );
```

```

DO_SQLBindCol( S0, 1, SQL_C_SLONG, 4, 0 );
DO_SQLBindCol( S0, 2, SQL_C_CHAR, 20, 0 );
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 10, 0, 4, 4 );

DO_LoadMem( S0, 1, "1", 4 );

strcpy(sql_statement, /* >> 0 << */
"SELECT KEYVAL, VARCHAR_COL FROM TEST_TABLE WHERE KEYVAL > {01}");
DO_substr(sql_statement, 1, "200" );
DO_SQLExecDirect( S0, sql_statement );

// Retrieve the data
DO_SQLFetch( 0 );
RR__printf( GetBindColumnData( 0, 1, 1 ) );
RR__printf( GetBindColumnData( 0, 2, 1 ) );
RR__printf( GetBindColumnData( 0, 1, 2 ) );
RR__printf( GetBindColumnData( 0, 2, 2 ) );
RR__printf( GetBindColumnData( 0, 1, 3 ) );
RR__printf( GetBindColumnData( 0, 2, 3 ) );
RR__printf( GetBindColumnData( 0, 1, 4 ) );
RR__printf( GetBindColumnData( 0, 2, 4 ) );
RR__printf( GetBindColumnData( 0, 1, 5 ) );
RR__printf( GetBindColumnData( 0, 2, 5 ) );

```

DO_SQLCancel

Cancels the processing of the present SQL statement.

This is rarely used within a script, although you could use it if only a subset of the rows are needed from a Select command.

Syntax

```
DO_SQLCancel( StatementIndex );
```

Return Value

None

Parameters

Parameter	Description
StatementIndex	Index into the table of ODBC statement handles.

Example

```
DO_SQLCancel( S0 );
```

DO_SQLCloseCursor

Closes a cursor associated with a handle and discards the results.

This cleanup function interacts minimally with do_odbc.

Syntax

```
DO_SQLCloseCursor( StatementIndex )
```

Parameters

Parameter	Description
-----------	-------------

StatementIndex	Index into the table of ODBC statement handles.
----------------	---

Example

```
DO_SQLFreeStmt( S0, SQL_DROP );
DO_SQLFreeStmt( S1, SQL_CLOSE );
DO_SQLSetStmtAttr( S1, SQL_ATTR_NOSCAN, SQL_NOSCAN_OFF, );
strcpy(sql_statement, /* >> 5 << */ "SELECT keyval, test_number, test_type FROM
dbo.test.table");
DO_SQLExecDirect( S1, sql_statement );
DO_SQLCloseCursor( S1);
DO_SQLFreeHandle(S1);
```

DO_SQLColAttribute

Returns descriptor information for a column in a result set.

In ODBC 3.x, this function replaces SQLColAttributes. SQLColAttribute returns descriptor information as a character string, a 32-bit descriptor-dependent value, or an integer value.

The SQLColAttribute replaces SQLColAttributes because it interacts with Descriptor information that was not present in prior versions of ODBC.

Syntax

```
DO_SQLColAttribute( StatementIndex, ColumnNum, ColumnAttribute, BufferLen )
```

Parameters

Parameter	Description
StatementIndex	Index into the table of ODBC statement handles.
ColumnNum	The column for the information.
ColumnAttribute	The attribute to be retrieved.
BufferLen	Amount of space for the information to be retrieved.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0);
DO_SQLBindParameter(S0, 1, SQL_PARAM_INPUT,
                    SQL_C_ULONG, SQL_INTEGER,
                    10, 0, "19", 4, 4 );
strcpy(sql_statement, /* >> 1 << */ "select
varchar_col, char_col, timestamp_col
from test_table where keyval < ?");
DO_SQLExecDirect( S0, sql_statement );
DO_SQLBindCol( S0, 1, SQL_C_CHAR, 50, 196658 );
DO_SQLBindCol( S0, 2, SQL_C_CHAR, 50, 50 );
DO_SQLBindCol( S0, 3, SQL_C_TIMESTAMP, 50, 2012741682 );
DO_SQLColAttribute( S0, 1, SQL_DESC_BASE_COLUMN_NAME, 50);
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLColumns

Retrieves the column information of the selected tables.

DO_SQLColumns is used to retrieve information from a series of columns within a table. Use DO_SQLNumResultsCols to sort through the result set.

Syntax

```
DO_SQLColumns( StatementIndex, QualifierName, TableOwner, TableName, ColumnName );
```

Return Value

The following information is retrieved for each matching column:

Column Name	Data Type	Comments
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier; NULL if not applicable to the data source.
TABLE_OWNER	Varchar(128)	Table owner identifier; NULL if not applicable to the data source.
TABLE_NAME	Varchar(128)	Table identifier.
COLUMN_NAME	Varchar(128)	Column identifier.
DATA_TYPE	Smallint	ODBC SQL data type.
TYPE_NAME	Varchar(128)	Data source-dependent data type name; for example, CHAR, VARCHAR, MONEY, LONG VARBINARY, or CHAR () for bit data.
PRECISION	Integer	Precision of the column on the data source.
LENGTH	Integer	Transfer size of the data. The length in bytes of data transferred on an SQLGetData or SQLFetch operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different than the size of the data stored on the data source. This value is the same as the PRECISION column for character or binary data.
SCALE	Smallint	Scale of the column on the data source.
RADIX	Smallint	Either 10 or 2. If it is 10, the values in PRECISION and SCALE give the number of decimal digits allowed for the column. If it is 2, the values in PRECISION and SCALE give the number of bits allowed in the column.
NULLABLE	Smallint	SQL_NO_NULLS if the column does not accept NULL values.
REMARKS	Varchar(254)	A description of the column.

Parameters

Parameter	Description
StatementIndex	Index into the table of ODBC statement handles.
QualifierName	Table qualifier identifier (accepts search patterns).
TableOwner	Name of the table owner (accepts search patterns).
TableName	Table name (accepts search patterns).
ColumnName	Column name to retrieve (accepts search patterns).

Example

```
DO_SQLAllocStmt( C0, S1 );
DO_SQLColumns( S1, "", "", "qctest", "" );
```

DO_SQLConnect

Performs a connection to the database.

The authorization string, if required by the database, must be present, since many drivers prompt the user for a password at runtime if the password is not present. This presents a problem when playing back multiple virtual users, as it is impractical for the test operator to respond to each prompt individually.

The call is for completeness only. QALoad translates a SQLConnect command into a SQLDriverConnect command. This facilitates the automatic detection of the Authorization string (password).

Syntax

```
DO_SQLConnect( ConnectionIndex, DSN, UDI, AuthStr );
```

Parameters

Parameter	Description
ConnectionIndex	Index into the table of connections.
DSN	Data source name.
UDI	User identifier.
AuthStr	Authorization string (password).

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "load", "db2", "sa" );
```

DO_SQLCopyDesc

Copies the fields of the source descriptor handle to the target descriptor handle.

If the values of the SourceDescHandle and TargetDescHandle parameters are associated with the same driver, the driver copies all descriptor fields. This is true even if the drivers are on different connections or environments.

If the values of the parameters are not associated with the same driver, only ODBC-defined fields are copied.

At this time QALoad does not store descriptor information. QALoad relies on ODBC to handle the calls and the descriptor data for the application.

Syntax

```
DO_SQLCopyDesc( SourceDescriptorHandleIndex, TargetDescriptorHandle );
```

Parameters

Parameter	Description
-----------	-------------

SourceDescriptorHandleIndex	The descriptor to copy over.
TargetDescriptorHandle	The descriptor receiving the copied information.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_DESC, C0, D1 );
DO_SQLAllocHandle( SQL_HANDLE_DESC, C0, D2 );
DO_SQLSetDescField( D1, 0, SQL_DESC_COUNT, 2, -6 );
DO_SQLSetDescRec( D1, 1, 4, 0, 4, 10, 0, 42919801, 1242388, 1242384 );
DO_SQLSetDescRec( D1, 2, 4, 0, 4, 10, 0, 42922201, 1242388, 1242384 );
DO_SQLCopyDesc( D1, D2 );
DO_SQLGetDescRec( D2, 1, 4 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D2 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D1 );
```

DO_SQLDescribeCol

Returns descriptor information to the statement handle.

The information is returned as a set of pointers describing the column name, column precision, column scale, and SQL type of the column. This information can be used as metadata for generic data handling.

Syntax

```
DO_SQLDescribeCol( StatementIndex, ColumnNumber, BufferLength )
```

Parameters

Parameter	Description
StatementIndex	The statement handle for the function call.
ColumnNumber	The number of the column in the table that SQLdescribeCol is retrieving information about. Column numbers start at 1 and advance from there.
BufferLength	The length of the buffer for the column name in bytes.

Example

```
DO_SQLFreeStmt( S0, SQL_CLOSE );
DO_SQLSetStmtAttr( S0, SQL_ATTR_NOSCAN, SQL_NOSCAN_OFF, );
strcpy(sql_statement, /* >> 3 << */ "SELECT * FROM dbo.test.table");
DO_SQLExecDirect( S0, sql_statement );
pcol = DO_SQLNumResultCols( S0 );
DO_SQLDescribeCol( S0, 1, 129);
DO_SQLColAttribute( S0, 1, SQL_DESC_AUTO_UNIQUE_VALUE, 0);
DO_SQLColAttribute( S0, 1, SQL_DESC_FIXED_PREC_SCALE, 0);
DO_SQLColAttribute( S0, 1, SQL_DESC_UPDATABLE, 0);
DO_SQLDescribeCol( S0, 2, 129);
DO_SQLColAttribute( S0, 2, SQL_DESC_AUTO_UNIQUE_VALUE, 0);
DO_SQLColAttribute( S0, 2, SQL_DESC_FIXED_PREC_SCALE, 0);
DO_SQLColAttribute( S0, 2, SQL_DESC_UPDATABLE, 0);
DO_SQLDescribeCol( S0, 3, 129);
DO_SQLColAttribute( S0, 3, SQL_DESC_AUTO_UNIQUE_VALUE, 0);
DO_SQLColAttribute( S0, 3, SQL_DESC_FIXED_PREC_SCALE, 0);
DO_SQLColAttribute( S0, 3, SQL_DESC_UPDATABLE, 0);
```

DO_SQLDisconnect

Closes the connection from the application to the database server.

Syntax

```
DO_SQLDisconnect( ConnectionIndex );
```

Parameters

Parameter	Description
ConnectionIndex	Index into the table of connections.

Example

```
DO_SQLDisconnect( C0 );
```

DO_SQLDriverConnect

Connects the application to the database.

Normally, the format of the connection string can vary between databases and ODBC drivers, but generally includes, at a minimum, the dataset name (DSN), user ID (UID), and password (PWD). If a password is required for the connection, it is important to include it in DO_SQLDriverConnect so the ODBC driver does not prompt the user at runtime for the connection string.

Syntax

```
DO_SQLDriverConnect( ConnectionIndex, ConnectionString );
```

Parameters

Parameter	Description
ConnectionIndex	Index into the table of connections.
ConnectionString	Complete ODBC connection string (see description).

Example

```
DO_SQLDriverConnect( C0, "DSN=Dan32;UID=dba;PWD=sql" );
```

DO_SQLEndTran

Provides the mechanism for all open transactions or all open transactions on a particular connection to be resolved.

Syntax

```
DO_SQLEndTran( nHandleType, nHandleIndex, nOperation );
```

Parameters

Parameter	Description
nHandleType	The type of Handle on which the transaction is being committed or rolled back: SQL_HANDLE_ENV, SQL_HANDLE_DBC.
nHandleIndex	Index into the table of statement or connection handles, or -666 which is

	used as a marker for the Environment handle.
nOperation	Either Commit the transaction or Roll it back (SQL_COMMIT or SQL_ROLLBACK).

Example

The following example commits all of the transactions open on connection index 1:

```
DO_SQLExecDirect( S3, sql_statement );
DO_SQLEndTran( SQL_HANDLE_DBC, C1, SQL_COMMIT );
```

In order to resolve all transactions, the call to DO_SQLEndTran has the value -666 as the handle index. This is a marker for the Environment handle.

```
DO_SQLEndTran( SQL_HANDLE_ENV, -666, SQL_COMMIT );
```

DO_SQLExecDirect

Prepares and executes a SQL statement.

Syntax

```
DO_SQLExecDirect( StatementIndex, SQLStatement );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
SQLStatement	SQL statement to be executed.

Example

```
DO_SQLExecDirect( S0, "Select * from emp_tutorial" );
```

DO_SQLExecute

Executes a prepared command using the current values of the parameter marker variables if any parameter markers exist in the command.

Syntax

```
DO_SQLExecute( StatementIndex );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.

Example

```
DO_SQLPrepare( S0, sql_statement );
DO_LoadMem( S0, 1, "17", 4 ); \
DO_LoadMem( S0, 2, "1234", 4 );
DO_LoadMem( S0, 3, "1235", 4 );
DO_LoadMem( S0, 4, "1236", 4 );
```

QALoad 5.02

```
DO_LoadMem( S0, 5, "1237", 4 );  
DO_SQLExecute( S0 );
```

DO_SQLFetch

Retrieves a single row of data.

Syntax

```
DO_SQLFetch( StatementIndex )
```

Parameters

Parameter	Description
StatementIndex	The index of the statement handle.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );  
strcpy(sql_statement, /* >> 1 << */ "SELECT MAX(keyval) FROM TESTDB.TEST_TABLE");  
DO_SQLExecDirect( S0, sql_statement );  
while (DO_SQLFetch( S0 ) != SQL_NO_DATA_FOUND )  
{  
    pReturnValue = DO_SQLGetData( S0, 1, SQL_C_LONG, 4 );  
    free(pReturnValue);  
}  
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLFreeConnect

Performs the cleanup of connection handles for ODBC/DB2 within a QALoad script.

The handle cleanup that was being performed in DO_SQLDisconnect is now being performed in DO_SQLFreeConnect or in DO_SQLFreeHandle.

Syntax

```
DO_SQLFreeConnect( ConnectionIndex );
```

Parameters

Parameter	Description
ConnectionIndex	Points to a structure that DB2/ODBC uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.

Example

```
DO_SQLAllocConnect( C0 );  
DO_SQLConnect( C0, "DB2RGR", "db2sa", "db2sa" );  
DO_SQLDisconnect( C0 );  
DO_SQLFreeConnect( C0 );
```

DO_SQLFreeHandle

In ODBC, DO_SQLFreeHandle handles statement and descriptor cleanup. In DB2, DO_SQLFreeHandle handles the additional cleanup of connection handles.

Each occurrence of DO_SQLFreeHandle must have a corresponding DO_SQLAllocHandle, either within the transaction loop or outside of the transaction loop.

DO_SQLFreeHandle replaces DO_SQLFreeStmt. Like DO_SQLAllocHandle, DO_SQLFreeHandle takes different parameters than its predecessor. DO_SQLFreeHandle is compatible with ODBC 3.x.

Syntax

```
DO_SQLFreeHandle( HandleType, HandleIndex )
```

Parameters

Parameter	Description
HandleType	Type of handle to be freed. Each handle has its own arguments: SQL_HANDLE_ENV, SQL_HANDLE_DBC (connection handle); SQL_HANDLE_STMT (statement handle); and SQL_HANDLE_DESC (descriptor handle).
HandleIndex	The address of the structure that ODBC should release from memory.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLBindCol( S0, 1, SQL_C_LONG, 4, 4 );
strcpy(sql_statement, /* >> 2 << */ "select MAX(keyval) from test_table");
DO_SQLExecDirect( S0, sql_statement );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLFreeStmt

Stops processing associated with a specific command_index and:

- ! Closes any open cursors associated with the command_index.
- ! Discards pending results.
- ! Frees all resources associated with command_index.

Consult your ODBC reference manual for details regarding the option parameter.

Syntax

```
DO_SQLFreeStmt( StatementIndex, Option );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
Option	One of the following predefined constants: SQL_CLOSE, SQL_DROP, SQL_UNBIND, SQL_RESET_PARAMS.

Example

```
DO_SQLFreeStmt( S0, SQL_DROP );
```

DO_SQLGetCursorName

Use on an open ODBC/DB2 statement to return a char * containing the cursor active on a particular statement.

This cursor can then be used in the execution of another query on another statement. Be aware that the pReturnValue must be freed by the script or a memory leak results.

Syntax

```
DO_SQLGetCursorName (StatementIndex, nBufferLength);
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
nBufferLength	The length of the buffer in bytes.

Example

In the following example, the pReturnValue is being placed in the sCursorName string immediately before the pReturnValue is freed.

```
char * DO_SQLGetCursor( <connection index>, <buffer length in bytes> );
```

An example on its correct use is as follows:

```
char sCursorName[19];
char *pReturnValue;
...
...
pReturnValue = DO_SQLGetCursorName( S1, 19 );
sprintf( sCursorName, "%s", pReturnValue );
free(pReturnValue);
strcpy(sql_statement, /* >> 3 << */ "UPDATE TESTDB.Test_Table set test_number = test_number
where current of ");
sprintf( sql_statement, "%s%s", sql_statement, sCursorName );
DO_SQLExecDirect( S2, sql_statement );
```

DO_SQLGetData

Retrieves data for a single column in the form of a string.

Call DO_SQLGetData after one or more rows have been retrieved from the result set by DO_SQLFetch. DO_SQLGetData allows large pieces of data to be returned by retrieving the data in parts if the variable length data is too large for a single call.

Syntax

```
DO_SQLGetData( nStmtIndex, nColNum, nCType, nBufLen )
```

Return Value

DO_SQLGetData can return the following values in the length/indicator buffer:

- ! Length of the data available to return
- ! SQL_NO_TOTAL
- ! SQL_NULL_DATA

Parameters

Parameter	Description
nStmtIndex	The index of the statement handle.
nColNum	The column number being returned.
nCType	The datatype.
nBufLen	The length of the buffer the data is returned in.

Example

```
strcpy(sql_statement, /* >> 1 << */ "SELECT MAX(keyval) FROM TESTDB.TEST_TABLE");
DO_SQLExecDirect( S0, sql_statement );
while (DO_SQLFetch( S0 ) != SQL_NO_DATA_FOUND )
{
pReturnValue = DO_SQLGetData( S0, 1, SQL_C_LONG, 4 );
free(pReturnValue);
}
```

DO_SQLGetDescField

Returns the value of a field of a descriptor record.

Use DO_SQLGetDescField to return the value of a descriptor record field. DO_SQLGetDescField can return the value of any field in any descriptor type. Make repeated calls to DO_SQLGetDescField to return settings from multiple fields of one or multiple descriptors in arbitrary order. DO_SQLGetDescField can also return driver-defined descriptor fields.

Syntax

```
DO_SQLGetDescField( DescriptorIndex, RecordNumber, FieldID, BufferLength, )
```

Parameters

Parameter	Description
DescriptorIndex	Points to the descriptor structure in memory.
RecordNumber	The record number of the descriptor structure to be retrieved.
FieldID	The field of the descriptor record to be retrieved.
BufferLen	The length of a character string or SQL_NTS being returned. SQL_LEN_BINARY_ATTR (macro) results if binary data is returned. SQL_IS_POINTER is Value, not binary or string data.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_DESC, C0, D0 );
strcpy(sql_statement, /* >> 1 << */ "UPDATE test_table SET integer_col = ? WHERE keyval = ?");
DO_SQLPrepare( S0, sql_statement );
DO_SQLSetDescRec( D0, 1, 4 );
DO_SQLSetDescRec( D0, 2, 4 );
DO_SQLGetDescField( D0, 1, SQL_DESC_CONCISE_TYPE, 261312 );
DO_SQLGetDescField( D0, 2, SQL_DESC_CONCISE_TYPE, 261312
```

DO_SQLGetDescRec

Returns the settings or values from fields of a descriptor record set by DO_SQLSetDescRec.

These fields include name, data type, and column or parameter data storage. Does not retrieve values for header fields.

To prevent the return of a setting, set the corresponding parameter to a null pointer.

Syntax

```
DO_SQLGetDescRec( nDescIndex, nRecordNumber, nBufLen );
```

Return Value

For a column or parameter, DO_SQLGetDescRec can retrieve the value of the following fields:

SQL_DESC_NAME

SQL_DESC_TYPE

SQL_DESC_OCTET_LENGTH

SQL_DESC_DATETIME_INTERVAL_CODE (types SQL_DATETIME and SQL_INTERVAL)

SQL_DESC_PRECISION

SQL_DESC_SCALE

SQL_DESC_NULLABLE

Parameters

Parameter	Description
nDescIndex	Points to the location of the descriptor structure in memory.
nRecordNumber	The descriptor record with fields to be set.
nBufLen	Descriptor record buffer length.

Example

```
DO_SQLSetDescRec( D1, 2, 4, 0, 4, 10, 0, 42922201, 1242388, 1242384 );
DO_SQLCopyDesc( D1, D2 );
DO_SQLGetDescRec( D2, 1, 4 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D2 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D1 );
```

DO_SQLGetEnvAttr

Gets a characteristic of an environment.

Syntax

```
DO_SQLGetEnvAttr( nAttribute, strAttrValue, nBufferLength, nStringLength );
```

Parameters

Parameter	Description
nAttribute	An environment attribute such as SQL_ATTR_CONNECTTYPE.
StrAttrValue	Current attribute value.
nBufferLength	Maximum size of attribute value.

nStringLength	Total number of bytes returned.
---------------	---------------------------------

Example

```
int rc = DO_SQLGetEnvAttr( SQL_ATTR_CONNECTTYPE, &connecttype, 0, NULL );
```

DO_SQLGetTypeInfo

Returns information about data types supported by the data source.

Syntax

```
DO_SQLGetTypeInfo( StatementIndex, SQLType );
```

Return Value

Data is returned as a result set with the following columns:

Column name	Data type
TYPE_NAME	Varchar(128)
DATA_TYPE	Smallint
PRECISION	Integer
LITERAL_PREFIX	Varchar(128)
LITERAL_SUFFIX	Varchar(128)
CREATE_PARAMS	Varchar(128)
NULLABLE	Smallint
CASE_SENSITIVE	Smallint
SEARCHABLE	Smallint
MONEY	Smallint
AUTO_INCREMENT	Smallint
LOCAL_TYPE_NAME	Varchar(128)

For details on the commands above, consult an ODBC reference manual.

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
SQLType (ODBC)	One of the following predefined constants: SQL_BIGINT, SQL_BINARY, SQL_BIT, SQL_CHAR, SQL_DATE, SQL_DECIMAL, SQL_DOUBLE, SQL_FLOAT, SQL_INTEGER, SQL_LONGVARBINARY, SQL_LONGVARCHAR, SQL_NUMERIC, SQL_REAL, SQL_SMALLINT, SQL_TIME, SQL_TIMESTAMP, SQL_TINYINT,

	SQL_VARBINARY, SQL_VARCHAR, SQL_ALL_TYPES
SQLType (DB2)	<p>One of the following predefined constants:</p> <p>SQL_BIGINT, SQL_BINARY, SQL_BLOB, SQL_BLOB_LOCATOR, SQL_CHAR, SQL_CLOB, SQL_CLOB_LOCATOR, SQL_DBCLOB, SQL_DBCLOB_LOCATOR, SQL_DECIMAL, SQL_DOUBLE, SQL_FLOAT, SQL_GRAPHIC, SQL_INTEGER, SQL_LONGVARBINARY, SQL_LONGVARCHAR, SQL_LONGVARGRAPHIC, SQL_NUMERIC, SQL_REAL, SQL_SMALLINT, SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_VARBINARY, SQL_VARCHAR, SQL_VARGRAPHIC.</p>

Example

```
DO_SQLAllocStmt( C0, S0 );
DO_SQLGetTypeInfo( S0, SQL_ALL_TYPES );
DO_SQLSetStmtOption( S0, SQL_ROWSET_SIZE, 16 );
DO_checkpoint( 9, 1 );
DO_SQLFreeStmt( S0, SQL_DROP );
```

DO_SQLNumResultCols

Determines the number of columns being returned in a result set.

This function returns an integer indicating the number of columns in the result set. Knowing the number of columns in the result set allows the application to use SQLDescribeCol and SQLColAttributes.

Syntax

```
DO_SQLNumResultCols( StatementIndex );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.

Example

```
int pcol = DO_SQLNumResultCols( S1 );
```

DO_SQLPutData

Use to place data into the database at run time.

You can also use this method to place data that is too large for a single bind. This method allows multiple calls to SQLPutData().

Syntax

```
DO_SQLPutData( int nStatementIndex, SQLPOINTER sData, long nIndLen);
```

Parameters

Parameter	Description
nStatementIndex	Index to the table of statement handles.

sData	The actual data in string form.
nIndLen	The length of the data.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
strcpy(sql_statement, "INSERT INTO dbo.TEST_TABLE (keyval, test_number, longvarchar_col)
VALUES (?, ?, ?)");
DO_SQLPrepare( S0, sql_statement );
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, 4, 0);
DO_SQLBindParameter( S0, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 16, 0, 0,
SQL_DATA_AT_EXEC);
DO_LoadMem( S0, 1, "26293", 0 );
DO_LoadMem( S0, 2, "9", 0 );
DO_SQLExecute( S0 );
DO_SQLParamData( S0 );
DO_SQLPutData( S0, "AAA", -3 );
DO_SQLParamData( S0 );
DO_SQLEndTran( SQL_HANDLE_DBC, C0, SQL_COMMIT );
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLPrepare

Prepares a SQL statement and associates the results with the command_index. The command is not executed until the DO_SQLExecute command is called.

Syntax

```
DO_SQLPrepare( StatementIndex, SQLString );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
SQLString	Text of the SQL statement to execute.

Example

```
char *sql_statement = "SELECT name from emp_tut";
DO_SQLPrepare( S1, sql_statement );
DO_SQLExecute ( S1 );
```

DO_SQLRetrieveParam Value

Retrieves a value of a SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT parameter, following the execution of the corresponding SQL statement.

Syntax

```
DO_SQLRetrieveParamValue( nStmtIndex, nParamNumber );
```

Parameters

Parameter	Description
nStmtIndex	Index into the table of ODBC/DB2 statement handles.

nParamNumber	Index of the parameter.
--------------	-------------------------

Example

```
char* sRow1 = NULL;
char* sRow2 = NULL;
char* sRow3 = NULL;
char* sRow4 = NULL;
DO_SQLBindParameter( S0, 1, SQL_PARAM_INPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 2, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 3, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 4, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
DO_SQLBindParameter( S0, 5, SQL_PARAM_OUTPUT, SQL_C_ULONG, SQL_INTEGER, 0, 0, 4, 4 );
strcpy( sql_statement, "{call setup_rows (?,?,?,?)}" );
DO_SQLPrepare( S0, sql_statement );
DO_LoadMem( S0, 1, "17", 4 );
DO_LoadMem( S0, 2, "1234", 4 );
DO_LoadMem( S0, 3, "1235", 4 );
DO_LoadMem( S0, 4, "1236", 4 );
DO_LoadMem( S0, 5, "1237", 4 );
DO_SQLExecute( s0 );
sRow1 = DO_SQLRetrieveParamValue( S0, 2 );
sRow2 = DO_SQLRetrieveParamValue( S0, 3 );
sRow3 = DO_SQLRetrieveParamValue( S0, 4 );
sRow4 = DO_SQLRetrieveParamValue( S0, 5 );
```

DO_SQLRowCount

Returns an integer indicating the number of rows affected by the last SQL statement associated with the specified command_index.

For inserts, updates, and deletes, the SQLRowCount value is available immediately after the command is executed. For Select commands, the value of this function depends on the capabilities of the specific ODBC driver used.

Syntax

```
DO_SQLRowCount( nStatementIndex );
```

Parameters

Parameter	Description
nStatementIndex	Index into the table of statement handles.

Example

```
DO_SQLExecDirect( S1, sql_statement );
int pcrow = DO_SQLRowCount( S1 );
DO_SQLFreeStmt( S1, SQL_CLOSE );
```

DO_SQLSetConnectAttr

Sets a characteristic of the connection.

Connection attributes are characteristics of the connection. They can be set before or after connecting. Connection attributes become part of the connect handle structure.

Syntax

```
DO_SQLSetConnectAttr( nConnectionIndex, nAttribute, nAttrValue, nStrLength );
```

Parameters

Parameter	Description
nConnectionIndex	Points to a structure that ODBC/DB2 uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.
nAttribute	For example: SQL_ATTR_AUTOCOMMIT is an attribute with set values SQL_TRUE or SQL_FALSE. Other connection attributes have different values. Note that QALoad does not allow SQL_ATTR_ENABLE_ASYNC to be true for ODBC. No asynchronous transactions will be handled.
nAttrValue	The value set for the attribute.
nStrLength	Can be a length pointer, or is ignored by ODBC and DB2.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLSetConnectAttr( C0, SQL_ATTR_AUTOCOMMIT,
"SQL_AUTOCOMMIT_ON", SQL_IS_INTEGER );
DO_SQLConnect( C0, "EDGARDO", "db2Admin", "db2Admin" );
DO_SQLAllocStmt( C0, S0 );
```

DO_SQLSetConnectOption

Sets options on the connection handle.

The following is a list of value option constants and the meanings of their respective value parameters.

Parameter	Value Type	Value Description
SQL_ACCESS_MODE	integer	Determines type of access this program uses.
SQL_AUTOCOMMIT	integer	0 = Autocommit off 1 = Autocommit on
SQL_LOGIN_TIMEOUT	integer	Number of seconds to wait for a login request to complete before returning to the application. The default is 15.
SQL_OPT_TRACE	integer	Integer value telling the Driver Manager whether or not to perform tracing. 0 = Tracing off (Default) 1 = Tracing on
SQL_OPT_TRACEFILE	string	Null-terminated character string containing the name of the trace file. If tracing is enabled, the Driver Manager writes to this file each time the application calls a function. If no trace file name is specified, the Driver Manager writes to SQL.LOG.
SQL_TRANSLATE_DLL	string	Null-terminated character string containing the name of a DLL containing the functions SQLClientToDataSource and

		SQLDataSourceToClient the driver loads and uses to perform tasks such as character set translation. This option may only be specified if the driver has connected to the data source.
SQL_TRANSLATE_OPTION	integer	32-bit flag value that is passed to the translate DLL. This option may only be specified if the driver has connected to the data source.
SQL_TXN_ISOLATION	integer	32-bit bitmask that sets the transaction isolation level for the current connection index. Refer to ODBC documentation for details on setting this parameter.

Syntax

```
DO_SQLSetConnectOption( ConnectionIndex, Option, Value );
```

Parameters

Parameter	Description
ConnectionIndex	Index into a table of ODBC connection handles.
Option	One of the valid option constants.
Value	Value associated with the option. Depending on the type of option used, it can either be NULL, an integer, or a pointer to a string.

Example

```
DO_SQLAllocStmt( C0, S0 );
DO_SQLSetStmtOption( S0, 1229, 0 );
DO_SQLSetStmtOption( S0, SQL_CONCURRENCY, 1);
DO_SQLSetConnectOption( C0, SQL_AUTOCOMMIT, 0 );
DO_SQLSetConnectOption( C0, SQL_TXN_ISOLATION, 1 );
DO_SQLFreeStmt( S0, SQL_CLOSE );
DO_SQLFreeStmt( S0, SQL_UNBIND );
```

DO_SQLSetCursorName

Associates a cursor name with an active command_index.

The only SQL statements that accept cursor names are UPDATE and DELETE.

Syntax

```
DO_SQLSetCursorName( StatementIndex, CursorName );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
CursorName	Name of the cursor.

Example

```
DO_SQLSetCursorName( S1, "C1" );
.....
```

```
.....
DO_SQLPrepare( S1,"UPDATE EMPLOYEE SET date=? WHERE CURRENT OF C1" );
```

DO_SQLSetDescField

Sets a descriptor field. A call to DO_SQLSetDescField can set a field of any descriptor type that can be set.

Call DO_SQLSetDescField first when dealing with an explicitly allocated descriptor, as it allocates the rows of an explicitly allocated descriptor.

Syntax

```
DO_SQLSetDescField( DescriptorHandle, RecordNumber, FieldID, Value, BufferLen )
```

Parameters

Parameter	Description
DescriptorHandle	Points to a structure used to describe rows of a result set, or a set of parameters to be bound to a statement.
RecordNumber	The record number of the descriptor.
FieldID	An attribute to set for the record.
Value	The value to set the FieldID to.
BufferLen	Length of the value coming in.

Example

```
DO_SQLSetDescField( D0, 0, SQL_DESC_COUNT, 2, -6 );
DO_SQLSetDescRec( D0, 1, SQL_INTEGER, 0, 4, 10, 0, "10", 4, 4 );
DO_SQLSetDescRec( D0, 2, SQL_INTEGER, 0, 4, 10, 0, "10", 4, 4 );
DO_SQLCopyDesc( D0, D1 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D1 );
DO_SQLGetDescRec( D2, 1, 0 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D2 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D1 );
```

DO_SQLSetDescRec

Sets multiple descriptor fields with a single call.

Use DO_SQLSetDescRec to change fields affecting the binding of a column or parameter without calling DO_SQLBindCol, DO_SQLBindParameter, or DO_SQLSetDescField. DO_SQLSetDescRec can set fields on a descriptor not currently associated with a statement, sets more fields than DO_SQLSetDescRec, can set fields on both an APD and an IPD in one call, and does not require a descriptor handle.

Because descriptors are associated with connections, a descriptor can carry over from one statement to the next and can be associated with different statements for continued bindings.

Syntax

```
DO_SQLSetDescRec( nDescIndex, nRecordNumber, nFieldId, nSubType, nLength, nPrecision,
nScale, pData, pStrLen, pIndicator )
```

Parameters

Parameter	Description
nDescIndex	Points to the location of the descriptor structure in memory.
nRecordNumber	The descriptor record with fields to be set.
nFieldId	The C data type of the field to be set.
nSubType	Applicable only for interval data types and for date and time data types, which have subtypes.
nLength	The length, in bytes, of a character string or binary datatype.
nPrecision	The maximum number of digits used by the column or parameter.
nScale	Scales the maximum number of digits to the right of the decimal point.
pData	Points to parameter or column value.
pStrLen	Points to a variable that will contain the total length in bytes of a dynamic argument.
pIndicator	Points to an indicator variable that contains SQL_NULL_DATA if the column value is a NULL. Otherwise the variable is 0.

Example

```
DO_SQLSetDescField( D0, 0, SQL_DESC_COUNT, 2, -6 );
DO_SQLSetDescRec( D0, 1, SQL_INTEGER, 0, 4, 10, 0, "10", 4, 4 );
DO_SQLSetDescRec( D0, 2, SQL_INTEGER, 0, 4, 10, 0, "10", 4, 4 );
DO_SQLCopyDesc( D0, D1 );
DO_SQLFreeHandle( SQL_HANDLE_DESC, D1 );
```

DO_SQLSetEnvAttr

Sets different aspects of the ODBC environment.

For ODBC 3.x, call this function immediately after calling DO_SQLAllocHandle to alert the environment handle as to which set of calls, ODBC 2.x or ODBC 3.x, the application will adhere.

Although QALoad captures and converts DO_SQLSetEnvAttr, you must enter the environment connection elements manually. The attributes must be set before the connection is allocated. Therefore, the calls to DO_SQLSetEnvAttr must be handled by script manipulation within pro_InitODBC, after the call to SQLSetEnvAttr made in this function.

DO_SQLSetEnvAttr can only be called if a connection handle is not allocated on the environment. Environment attributes set by the application persist until DO_SQLFreeHandle is called on the environment.

Syntax

```
DO_SQLSetEnvAttr(Attribute, AttributeValue, Indicator)
```

Parameters

Parameter	Description
-----------	-------------

Attribute	The specific property the application is setting.
AttributeValue	The value of the specific property that the application is setting.
Indicator	Can be a length pointer, or is ignored by ODBC/DB2.

Example

```
DO_SQLAllocHandle( SQL_HANDLE_ENV, 0, c0 );
DO_SQLSetEnvAttr( SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC2, -6 );
DO_SQLAllocHandle( SQL_HANDLE_STMT, c0, s0 );
```

DO_SQLSetPos

Positions a cursor within a retrieved block of data.

Syntax

```
DO_SQLSetPos( StatementIndex, nRow, nRefresh, nLock );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
nRow	Absolute position of the cursor within the retrieved block of data. nRow must be a value from 1 to the number of rows in the rowset.
nRefresh	Specifies whether or not to refresh the buffer value for the row specified by nRow. If TRUE (1), the driver refreshes the buffer value. If FALSE (0), the driver does not change the buffer value.
nLock	Specifies whether or not to lock the row for subsequent update operation. If TRUE (1), the driver requests a lock for the row. If FALSE (0), the driver applies the form of concurrency control specified in a call to DO_SQLSetScrollOptions.

Example

```
int iRow = 1;
DO_SQLSetPos( S1, iRow, FALSE, FALSE );
```

DO_SQLSetScrollOptions

Allows the application to specify the type of cursor behavior in three areas: concurrency control, sensitivity to changes made by other transactions, and rowset size.

Syntax

```
DO_SQLSetScrollOptions( int StmtIndex, UWORD fConcurrency, SDWORD crowKeyset, UWORD crowRowset );
```

Parameters

Parameter	Description
StmtIndex	Index to the table of ODBC/DB2 statement handles.

fConcurrency	Specifies concurrency control for the cursor.
crowKeyset	Number of rows for which to buffer keys.
crowRowset	Number of rows in a rowset.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloadb2", "sa", "" );
DO_SQLAllocStmt( C0, S0 );
DO_SQLSetScrollOptions( S0, SQL_CONCUR_LOCK, SQL_SCROLL_STATIC, 0 );
```

DO_SQLSetStmtAttr

Sets statement attributes and, as a result, sets descriptor fields.

When calling DO_SQLSetStmtAttr to set fields, rather than DO_SQLSetDescField, it is not necessary to obtain a descriptor handle for the function call.

When using DO_SQLSetStmtAttr, calling it for a statement can affect other statements if the statement's Application Parameter Descriptor (APD) or Application Row Descriptor (ARD) is explicitly allocated and associated with other statements.

DO_SQLSetStmtAttr modifies the APD or ARD and those modifications apply to all statements with which the descriptor is associated. To avoid this, disassociate the descriptor from the other statement using DO_SQLSetStmtAttr to change the descriptor handle of SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC. Then call DO_SQLSetStmtAttr again.

When setting a statement attribute also sets a descriptor field, the field is set only for the descriptors currently associated with the statement identified by the StatementHandle parameter. Subsequent attribute settings are not affected. When DO_SQLSetDescField sets a descriptor field that is also a statement attribute, it also sets the corresponding statement attribute.

Syntax

```
DO_SQLSetStmtAttr( int nStmtIndex, long nAttribute, long nAttrValue, long nStrLength );
```

Parameters

Parameter	Description
nStmtIndex	Points to a structure that ODBC uses to track different statement settings and descriptor settings within the same connection handle as the statement.
nAttribute	Attribute. For example: SQL_ATTR_APP_ROW_DESC. Note that even at the statement level, QALoad does not permit asynchronous transactions.
nAttrValue	The value set for the attribute.
nStrLength	Attribute length

Example

```
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLAllocHandle( SQL_HANDLE_DESC, C0, D0 );
DO_SQLSetStmtAttr( S0, SQL_ATTR_APP_PARAM_DESC, D0, SQL_IS_POINTER );
DO_SQLSetDescField( D0, 0, SQL_DESC_COUNT, 2, -6 );
```


DO_SQLSetStmtOption

Sets the boundaries of a specific statement handle.

Following is a list of value option constants and the meanings of their respective value parameters.

Parameter	Description
SQLBindType	<p>A 32-bit value that sets the binding orientation used when <code>DO_SQLExtendedFetch</code> is called on the associated C.</p> <p>Column-wise binding is selected by supplying the defined constant <code>SQL_BIND_BY_COLUMN</code> for the argument <code>vParam</code>.</p> <p>Row-wise binding is selected by supplying a value for <code>vParam</code> specifying the length of a structure or an instance of a buffer into which result columns will be bound.</p> <p>The length specified in <code>vParam</code> must include space for all of the bound columns and any padding of the structure or buffer. This ensures that when the address of a bound column is incremented with the specified length, the result points to the beginning of the same column in the next row. When using the size of operator with structures or unions in ANSI C, this behavior is guaranteed.</p> <p>Column-wise binding is the default binding orientation for <code>DO_SQLExtendedFetch</code>.</p>
SQLMaxLength	A value corresponding to the maximum amount of data that can be retrieved from a single column with a <code>LONG VARCHAR</code> or <code>LONG VARBINARY</code> data type.
SQLMaxRows	A value corresponding to the maximum number of rows to return to the application for a <code>SELECT</code> command. If <code>vParam</code> equals 0 (Default), the driver returns all rows.
SQLNoScan	A value. If <code>TRUE</code> (1), the driver does not scan SQL strings for escape clauses. Instead, the driver sends the command directly to the data source. If <code>FALSE</code> (Default, value 0), the driver scans for escape clauses.
SQLQueryTimeout	A value corresponding to the number of seconds to wait for an SQL statement to execute before returning to the application. If <code>vParam</code> equals 0 (Default), there is no time out.

Syntax

```
DO_SQLSetStmtOption( StatementIndex, nOption, nParam );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
nOption	One of the valid option constants.
nParam	One of the constants listed below.

Example

```
DO_SQLAllocStmt( C0, S0 );
DO_SQLSetStmtOption( S0, SQL_QUERY_TIMEOUT, 60 );
DO_SQLSetStmtOption( S0, SQL_ASYNC_ENABLE, 1 );
```

DO_SQLSpecialColumns

Retrieves information about columns within a specified table.

DO_SQLSpecialColumns retrieves the following information:

- ! The optimal set of columns that uniquely identifies a row in the table.
- ! Columns that are automatically updated when any value in the row is updated by a transaction.
- ! The data is returned as a result set.

Syntax

```
DO_SQLSpecialColumns( StatementIndex, fColType, szTableQualifier, szTableOwner, szTableName,
fScope, fNullable );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
fColType	Type of column to return. Must be one of the following values: SQL_BEST_ROWID : Returns the optimal column or set of columns that, by retrieving values from the column or columns, allows any row in the specified table to be uniquely identified. A column can be either a pseudo column specifically designed for this purpose (as in ODBC ROWID or Ingres TID) or the column or columns of any unique index for the table. SQL_ROWVER : Returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction (as in SQLBase ROWID or Sybase TIMESTAMP).
SzTableQualifier	Qualifier name for the table.
SzTableOwner	Owner name for the table.
SzTableName	Table name.
fScope	Minimum required scope of the ROWID. The returned ROWID may be of greater scope. Must be one of the following: SQL_SCOPE_CURROW : The ROWID is guaranteed to be valid only while positioned on that row. A later reselect using ROWID may not return a row if the row was updated or deleted by another transaction. SQL_SCOPE_TRANSACTION : The ROWID is guaranteed to be valid for the duration of the current transaction. SQL_SCOPE_SESSION : The ROWID is guaranteed to be valid for the duration of the session (across transaction boundaries).
fNullable	Determines whether to return special columns that have NULL values. Must be one of the following: SQL_NO_NULLS : Exclude special columns that can have NULL values. SQL_NULLABLE : Return special columns even if they can have NULL values.

Example

```
DO_SQLSpecialColumns( S1, SQL_ROWVER, "", "", "qc_test", SQL_SCOPE_TRANSACTION, SQL_NULLABLE
);
int pcol = DO_SQLNumResultCols( S1 );
```

DO_SQLStatistics

Retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.

Syntax

```
DO_SQLStatistics( StatementIndex, szTableQualifier, szTableOwner, szTableName, fUnique, fAccuracy );
```

Parameter

Parameter	Description
StatementIndex	Index into the table of statement handles.
SzTableQualifier	Qualifier name.
SzTableOwner	Owner name.
SzTableName	Table name.
fUnique	Type of index: SQL_INDEX_UNIQUE or SQL_INDEX_ALL.
fAccuracy	Importance of the CARDINALITY and PAGES columns in result set: SQL_ENSURE: Requests that the driver unconditionally retrieves the statistics. SQL_QUICK: Requests that the driver retrieves results only if they are readily available from the server. In this case, the driver does not ensure the values are current.

Example

```
DO_SQLStatistics( S0, "", "", "qc_test", SQL_INDEX_ALL, SQL_QUICK );
```

DO_SQLTables

Returns the list of table names stored in a specific data source. The driver returns the information as a result set.

The szTableQualifier, szTableOwner, and szTableName parameters accept search patterns. Refer to ODBC documentation regarding the use of search patterns.

Syntax

```
DO_SQLTables( StatementIndex, szTableQualifier, szTableOwner, szTableName, szTableType );
```

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
SzTableQualifier	Qualifier name.
SzTableOwner	Owner name.
SzTableName	Table name.
SzTableType	List of table types to match. If szTableType is not an empty string, it must

	contain a list of comma-separated, single quoted values for the types of interest (for example: TABLE or VIEW). Valid table type identifiers may include TABLE, VIEW SYSTEM TABLE, ALIAS, SYNONYM, or other data source-specific identifiers.
--	---

Example

```
DO_SQLTables( S0, "", "", "", " 'TABLE', 'VIEW', 'SYSTEM TABLE', 'ALIAS', 'SYNONYM' " );
```

DO_SQLTransact

Requests a commit or rollback operation for all update, insert, and delete transactions in progress on all command indexes associated with a connection.

Can also request that a commit or rollback operation be performed for all connections by specifying a connection index of -1.

Syntax

```
DO_SQLTransact( ConnectionIndex, fType );
```

Parameters

Parameter	Description
ConnectionIndex	Index into a table of ODBC connection handles or -1 to indicate the operation should be performed on all connections.
fType	One of the following two constants: SQL_COMMIT or SQL_ROLLBACK.

Example

```
DO_SQLFreeStmt( C1, SQL_DROP );
DO_SQLTransact( S0, SQL_COMMIT );
```

DO_substr

Finds a value within a string.

Syntax

```
DO_Substr( string, placeholder, value );
```

Parameters

Parameter	Description
String	The string to be searched.
Placeholder	Location in the string.
Value	The token to search for.

Example

```
DO_SQLAllocStmt( C0, S0 );
DO_SQLSetStmtOption( S0, SQL_QUERY_TIMEOUT, 60 );
strcpy(sql_statement, "SELECT {1}, {2}, {3} FROM Customers");
DO_substr(sql_statement, 1, "CustomerID" );
```

```
DO_substr(sql_statement, 2, "CompanyName" );
DO_substr(sql_statement, 3, "ContactName" );
DO_SQLExecDirect( S0, sql_statement );
DO_SQLFreeStmt( S0, SQL_CLOSE );
```

GetBindColumnData

Retrieves data from one of the rows that are returned by [DO_SQLFetch](#) calls, after a combination of [DO_SQLSetStmtAttr](#) and [DO_SQLBindCol](#) calls.

Syntax

```
GetBindColumnData (int nIndex, int nColumn, int nRow);
```

Parameters

Parameter	Description
nIndex	The statement index.
nColumn	The column of data to return.
nRow	The row of data to return.

Returns

char* containing the data or an error

Example

```
DO_SQLFetch( S0 );
RR__printf( GetBindColumnData( S0, 1, 1 ) );
RR__printf( GetBindColumnData( S0, 2, 1 ) );
RR__printf( GetBindColumnData( S0, 1, 2 ) );
RR__printf( GetBindColumnData( S0, 2, 2 ) );
RR__printf( GetBindColumnData( S0, 1, 3 ) );
RR__printf( GetBindColumnData( S0, 2, 3 ) );
RR__printf( GetBindColumnData( S0, 1, 4 ) );
RR__printf( GetBindColumnData( S0, 2, 4 ) );
RR__printf( GetBindColumnData( S0, 1, 5 ) );
RR__printf( GetBindColumnData( S0, 2, 5 ) );
```

Oracle 7

Oracle 7 Index

DO_autocommitoff

Disables the automatic commit of every SQL data manipulation command.

DO_autocommiton

Enables the automatic commit of every SQL data manipulation command.

DO_binddate

Binds a date variable to a bind variable in a SQL statement.

DO_BindForUpdateRowID

Binds a rowid in an UPDATE or DELETE statement where the rowid originates from a previous SELECT FOR

UPDATE statement.

[DO_bindnull](#)

Binds a NULL value to a bind variable in a SQL statement.

[DO_bindstring](#)

Binds a program variable to a bind variable in a SQL statement.

[DO_BindV](#)

Binds a program variable to a bind variable in a SQL statement.

[DO_cleanup](#)

Deallocates cursors, logon structures and allocated memory when the script is aborted.

[DO_commit](#)

Commits the current transaction.

[DO_FetchIters](#)

Identifies the number of fetch iterations that will be applied to the succeeding fetch loop. The Fetch Loop will execute (n) times according to the fetch iteration value.

[DO_get_select_variable](#)

Places the select-list item data recently fetched by DO_process_select_list in a program variable, which may then be used in subsequent statements.

[DO_GetSelectData](#)

Copies the data retrieved from a SQL SELECT statement into a program variable.

[DO_init_alen](#)

A routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_ScalarBindA.

[DO_init_data](#)

A routine that allocates and initializes all of the logon data areas, cursor data areas, structures, and so on which are used to run the script.

[DO_init_indp](#)

A routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_Bindv and DO_ScalarBindA calls.

[DO_oclose](#)

Disconnects a previously opened cursor, returning all resources back to the Oracle server.

[DO_oexec](#)

Executes the SQL statement associated with a cursor.

[DO_olog](#)

Establishes a connection between QALoad and an Oracle database.

[DO_ologof](#)

Closes a connection to the Oracle server, freeing its resources.

[DO_oopen](#)

Opens a cursor to the database.

DO_oopt

Sets rollback options for non-fatal errors on multi-row INSERT and UPDATE SQL statements and determines whether to wait for requested resources or return errors.

DO_oparse

Parses a SQL statement or a PL/SQL block and associates it with a cursor index.

DO_process_select_list

Fetches select-list data from the Oracle database. It is generally called repeatedly until there are no more rows satisfying the SQL select request.

DO_rollback

Rolls back the current transaction.

DO_ScalarBindA

Binds a program variable to a bind variable in a SQL statement.

DO_SoftClose

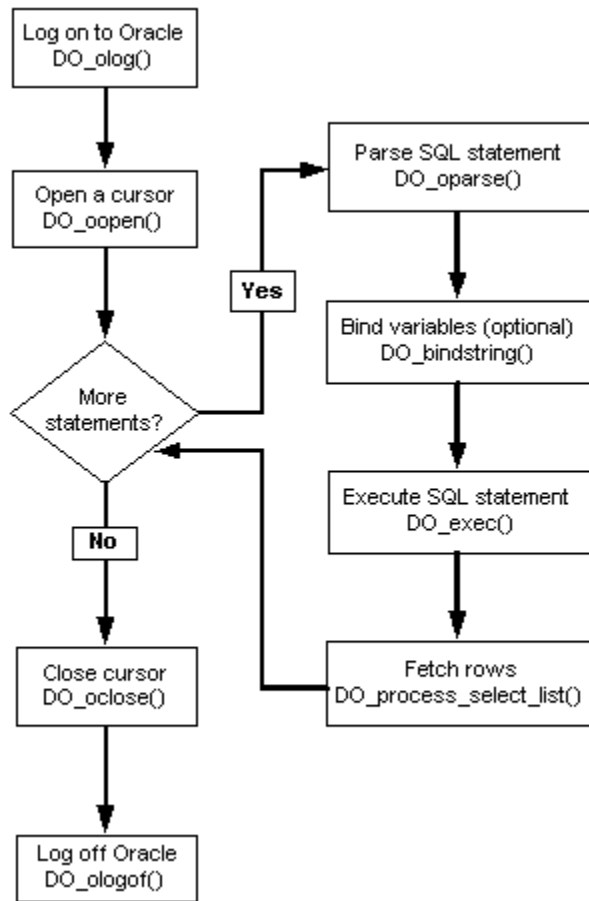
Closes a cursor without destroying its resources on the server.

Oracle 7 cursors

An Oracle cursor is a handle to a region in the System Global Area on the server where parsed SQL statements and other information for processing are kept. In Oracle applications, such as a QALoad script, this region is mapped to OCI commands through a data structure called the Cursor Data Area (CDA). Cursor Data Areas are opened and closed by an application and their addresses are passed to those OCI commands which are used to manipulate SQL statements. In QALoad, Cursor Data Areas are managed internally but are referenced by constants (for example, CDA(0), CDA(1), etc.) that represent indexes into a table of CDAs. Cursor indexes are used to open and close cursors and are then passed to the appropriate Oracle QALoad script commands.

Oracle 7 command sequence

The following figure shows the standard sequence of commands for a QALoad Oracle 7-based script.



Adding QALoad script commands to perform an SQL statement

1. Create the appropriate handles and increment the `HANDLE_COUNT` parameter in the QALoad script.
 - a. Find the number after `HANDLE_COUNT` in the QALoad script. Note the current number, and then add one (1) to this number. One new handle will be allocated for this SQL statement example. For example, if the following line is in the script:
`HANDLE_COUNT 39`
 edit it to read:
`HANDLE_COUNT 40`
 - b. Then, allocate a statement handle (note that you can use a pre-allocated error handle; this error handle is usually created and freed during the logon and logoff sequences).
 - c. Associate the number 39 with this new handle to be allocated, as shown below:
`DO_OCISHandleAlloc(HNDL(39), OCI_HTYPE_STMT);`
2. Prepare the SQL statement. Use the new allocated statement handle in a call to `DO_OCISstmtPrepare`, as shown below. Note that the SQL statement is enclosed in double-quotes. The last parameter, `OCI_NTV_SYNTAX`, will allow the Oracle database to parse the SQL statement based on its native parsing syntax. This is the recommended parameter setting for this call.
`DO_OCISstmtPrepare(HNDL(39), "SELECT * FROM emp", OCI_NTV_SYNTAX);`
3. Bind all input, return, and PL/SQL stored procedure/function parameters. For each input parameter, return parameter, or PL/SQL stored procedure/function parameter, a bind variable must be defined to specify the data type of the bound value, the input value as a variable or constant, and a buffer area to receive any changes to the value on output (as from a PL/SQL stored procedure/function INPUT/OUTPUT variable). See [DO_OCIBind](#) for more information.
`DO_OCIBind(HNDL(6), HNDL(1), ":VPKEY", _INTEGER, 4, NULL, NULL, (ub1 *) "65", (ub1 *) &INTEGER_6_VPKEY_1);`

- Execute the SQL statement. Once all the bind statements are added, the SQL statement execution statement is now added. Below is a typical call to `DO_OCISmtExecute`. Note that the `iters` parameter (the 2nd parameter) should be set to 1 in most cases.

```
DO_OCISmtExecute( HNDL(6), 1, OCI_DEFAULT );
```

- Fetch the data in a loop with `DO_OCIProcessSelectList`. If the SQL statement is a `SELECT` statement with one or many returning values, you will need to add a `DO_OCIProcessSelectList` loop. This will iterate through the fetched data until there is no returned data. This loop should be present even if no fetched values are retrieved into variables (see Step 6). Note that the `fetchcount` parameter (the 2nd parameter) should be set to 1 to retrieve a single row at a time.

```
while ( DO_OCIProcessSelectList( HNDL(6), 1 ) )
{
    /* Here is where the calls to retrieve individual fetched data
       items to variables using DO_OCISGetSelectData
       is put (see Step 6) */
} /* end of DO_process_select_list */
```

- Retrieve individual fetched data values to variables in the `DO_OCIProcessSelectList` loop with `DO_OCISGetSelectData`. If there are individual fetched items whose values are to be put into C-style variables for later use, a `DO_OCISGetSelectData` call should be placed within the brackets of the while statement for each value to be retrieved. Note that the fetch number, column number, and row number of the desired fetch data must be known, and a buffer variable for the data must be created. The sample code below shows a call to `DO_OCISGetSelectData` to retrieve the value returned from the first column and first row of the first fetch into the address of the variable `ENO`.

```
while ( DO_OCIProcessSelectList( HNDL(6), 1 ) )
{
    DO_OCISGetSelectData(HNDL(6), 1, 1, 1, &ENO, _NONE_ , NULL, 0);
}
```

- Commit the SQL statement. Once the SQL statement is finished processing, it can be committed. This commit can be done any time after the statement has executed until the service context handle is de-allocated (usually during the Oracle logoff sequence).

```
DO_OCISCommit( HNDL(3), HNDL(1), 0 );
```

- Once the statement has fetched all its data after execution, the statement handle should be freed with a call to `DO_OCISHandleFree` for the statement handle.

```
DO_OCISHandleFree( HNDL(6), OCI_HTYPE_STMT);
```

DO_autocommitoff

Disables the automatic commit of every SQL data manipulation command.

Syntax

```
DO_autocommitoff(LDAIndex );
```

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

`ocof`

Example

```
DO_autocommitoff( LDA(0) );
```

DO_autocommiton

Enables the automatic commit of every SQL data manipulation command.

Syntax

```
DO_autocommiton( LDAIndex );
```

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

ocon

Example

```
DO_autocommiton( LDA(0) );
```

DO_binddate

Binds a date variable to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:).

DO_binddate commands must be placed between the DO_oparse and the DO_oexec commands. To bind by position, as opposed to by name, preface the position with an @symbol.

Syntax

```
DO_binddate( cursor, name, &oradate_structure );
```

Parameters

Parameter	Description
cursor	Cursor table index.
name	Pointer to the name of the bind variable (null terminated string).
oradate_structure	An ORADATE structure. This structure is defined in the header files provided by Oracle. The '&' is the C address operator which specifies that the address or pointer to an ORADATE structure is being passed. year: Four-digit year. month: Two-digit month. day: Two-digit day of the month. hour: Hour in the day (0 to 23). min: Minute within the hour (0-59). second: Seconds within the minute (0-59).

Equivalent OCI

obndrv

Example

This example shows a **Select** command with one bind variable:

```
/* ORADATE declarations follow */
ORADATE DATE_0_6;
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE HIRE_DATE >:hiredate " );
DO_makedate( &DATE_0_6, 1980, 12, 17, 0, 0, 0 );
DO_binddate( CDA(0), ":hiredate", &DATE_0_6 );
DO_oexec( CDA(0) );
```

DO_BindForUpdateRowID

Binds a rowid in an UPDATE or DELETE statement where the rowid originates from a previous SELECT FOR UPDATE statement.

Syntax

```
DO_BindForUpdateRowID( cursor1, cursor0, bind_variable_name );
```

Parameters

Parameter	Description
cursor1	Cursor index.
cursor0	Cursor index of the FOR UPDATE statement.
bind_variable	Name of a rowid bind variable.

Example

```
DO_oopen( LDA(0), CDA(0) );
DO_oparse( CDA(0), "SELECT EMPNO, ENAME FROM EMP FOR UPDATE OF EMPNO, ENAME");
DO_oexec( CDA(0) );

n = DO_process_select_list( CDA(0), 3 );

DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1), "UPDATE EMP SET EMPNO=:empno, ENAME=:ename WHERE ROWID = :row_id");
DO_BindForUpdateRowID( CDA(1), CDA(0), ":row_id" );

DO_BindV(CDA(1), (text*):empno",_VARCHAR2,
           4, NULL, (ub1 *) "7421",
           (ub1 *) VARCHAR2_0_empno_0);
DO_BindV(CDA(1), (text*):ename",_VARCHAR2,
           4, NULL, (ub1 *) "WARD",
           (ub1 *) VARCHAR2_0_ename_1);
DO_oexec( CDA(1) );
```

DO_bindnull

Binds a NULL value to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). The input value will be set null. Bind statements must be placed after the DO_oparse and before the DO_oexec.

DO_bindnull cannot be used with bind variables that return results, as in stored procedures OUTPUT parameters.

Syntax

```
DO_bindnull( cursor, name );
```

Parameters

Parameter	Description
cursor	Cursor table index.
name	The name of the bind variable as a null-terminated character string.

Equivalent OCI

obndrv, obndrn

Example

This example shows a Select command with two bind variables, :empid and :id, which are being bound to a NULL value.


```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID =:empid AND EMP_DEPT_ID =
:id " );
DO_bindnull( CDA(0), ":empid" );
DO_bindnull( CDA(0), ":id" );
DO_oexec( CDA(0) );
```

DO_bindstring

Binds a program variable to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_bindstring must be called after the DO_oparse and before the DO_oexec. Once you have bound a variable, you can change the value and length and then call another DO_oexec.

DO_bindstring only supports the binding of strings, nulls, and dates. If you need to bind a numeric value, convert it first to a string before passing it to DO_bindstring. If needed, Oracle automatically converts character data types to numeric.

 **Note:** DO_bindstring is a deprecated command. Use DO_BindV or DO_ScalarBindA. DO_bindstring binds every data type as a fixed character and forces the Oracle server to make implicit database conversions. Also, you must variabilize OUTPUT variables or they will overwrite the input data held by string constants.

Syntax

```
DO_bindstring( cursor, name, value );
```

Parameters

Parameter	Description
cursor	Cursor table index.
name	Pointer to the name of the bind variable (null terminated).
value	Pointer to a string containing the value for the bind variable (null terminated).

Equivalent OCI

obndrv, obndrn

Example

This example shows a **Select** command with two bind variables, :empid and :id.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID =:empid AND EMP_DEPT_ID =
:id " );
DO_bindstring( CDA(0), ":empid", "200" );
DO_bindstring( CDA(0), ":id", "100" );
DO_oexec( CDA(0) );
```

DO_BindV

Binds a program variable to a bind variable in a SQL statement.

A DO_BindV is generated wherever an obndrv occurred in the capture file. DO_BindV accurately reproduces the original bind call made by the application. This eliminates extra data conversion steps and improves handling of OUTPUT variables to Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:). DO_BindV must be called after the DO_oparse and before a DO_oexec. Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable.

Currently, DO_BindV is not supported for cursor, mlabel, packed-decimal, olabel, PCC-descriptor, and the new Oracle 8 datatypes.

Syntax

```
DO_BindV(index, name, type, progv1, indp, input, progv);
```

Parameters

Parameter	Description
index	Cursor table index.
name	Pointer to name of the bind variable (null terminated).
type	External datatype of bind variable.
progv1	Size of progv. This is the maximum size of the buffer. If binding an OUTPUT variable, progv1 must be at least as large as the expected output value.
indp	Pointer to a null indicator variable: pIndp[0] points to make_indp[0], and make_indp holds the value of the null indicator. If make_indp[0]=SET_NULL, the input will be passed to Oracle as null. Otherwise, data is passed as shown in the bind call.
input	Pointer to buffer containing input data.
progv	Output data buffer.

Equivalent OCI

obndrv, obndrn


Example

This example shows a `Select` command with two bind variables, `:empid` and `:id`.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID = :empid AND EMP_DEPT_ID = :id" );
make_indp[0]=0
DO_BindV(CDA(0), ":empid", _STRING, 48, pIndp[0], "200", STRING_0_empid_3);
make_indp[1]=0
DO_BindV (CDA(0), ":id", _STRING, 48, pIndp[1], "100", STRING_0_id_3);
DO_oexec ( CDA(0) );
```

DO_cleanup

Deallocates cursors, logon structures and allocated memory when the script is aborted.

 **Note:** Do not modify or move this command.

Syntax

```
sword DO_cleanup();
```

Parameters

None.

DO_commit

Commits the current transaction.

Syntax

```
DO_commit( LDAIndex );
```

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

```
ocom
```

Example

```
DO_commit( LDA(0) );
```

DO_FetchIters

Identifies the number of fetch iterations that will be applied to the succeeding fetch loop. The Fetch Loop will execute (n) times according to the fetch iteration value.

The fetch iteration value is derived from the script capture's fetch iteration data for each `Select` statement. However, the Fetch Iteration Override in Oracle Convert Options may be used to replace all fetch iteration values in the script. The override range is 1-1000000. The default value for each convert activity is 0 (no override).

Syntax

```
DO_FetchIters( cursorIndex, fetchIterationValue );
```

or

```
DO_FetchIters( statementHandleIndex, fetchIteration Value );
```

Parameters

Parameter	Description
cursorIndex	An index to an allocated OC17 cursor or Oracle 8 statement handle used in the previous call to oparse, osql3, upipse, upiosq, or OCISmtPrepare.
fetchIterationValue	Number of iterations to be applied to the succeeding fetch-loop.
statementHandleIndex	An index to an allocated OC17 cursor or Oracle 8 statement handle used in the previous call to oparse, osql3, upipse, upiosq, or OCISmtPrepare.

Example

The following OC17 example shows how DO_FetchIters is used relative to the parse and fetch-loop:

```
DO_oparse( CDA(0), "select ename, empno, mgr from emp" );
DO_FetchIters( CDA(0), ITERS(4) );
DO_oexec( CDA(0) );
```

```
while (DO_process_select_list( CDA(0), 1 ))
//1 = the number of rows per iteration
{
}
```


The following OCI18 example shows how DO_FetchIters is used relative to the prepare and fetch-loop.

```
DO_OCISmtPrepare( HNDL(6), "select empno from emp", OCI_NTV_SYNTAX );
DO_FetchIters( HNDL(6), ITERS(13) );
DO_OCIDefine( HNDL(6), HNDL(1), 1, 1, _VARCHAR2, 4, IS_ATTRIBUTE );
DO_OCISmtExecute( HNDL(8), HNDL(6), HNDL(1), 1, OCI_DEFAULT );

while (DO_OCIProcessSelectList( HNDL(6), 1 ))
//1 = the number of rows per iteration
{
DO_OCI8GetSelectData(FETCH(1), COL(1), ROW(1), &FD_stmnt_1_col_1_row_1, _NONE_, "", 0 );
}
```

DO_get_select_variable

Places the select-list item data recently fetched by DO_process_select_list in a program variable, which may then be used in subsequent statements.

 **Note:** DO_process_select_list is called repeatedly in a loop. Each call to DO_process_select_list fetches a number of rows into the script's internal buffer. The number of rows is specified in the second parameter of DO_process_select_list. DO_get_select_variable, in turn, copies the fetched data from a specific row and the select-list item into a program variable.

All select list items are converted by the server into a null terminated string format prior to being processed by your script. Therefore, dates and numbers appear as readable ASCII character strings.

The program does not check to verify that the length of the buffer is sufficiently large to contain the returned value.

Syntax

```
DO_get_select_variable( pos, row, value );
```

Parameters

Parameter	Description
pos	Variable to retrieve (starts at 1).
row	Row in the buffer to retrieve (starts at 1).
value	Pointer to a character array into which the data is placed.

Example


This example shows how the first select-list item from the second fetched row is copied to the program variable coname.

```
char coname[128];
:
:
DO_oexec( CDA(0) ); /* Exec for statement 2 */
while ( DO_process_select_list( CDA(0), 30 ) )
{
DO_get_select_variable( 1, 2, coname );
}
```

DO_GetSelectData

Copies the data retrieved from a SQL SELECT statement into a program variable.

DO_GetSelectData processes the data retrieved by DO_process_select_list() by copying the value of the fetched data to another program variable. Typically, the program variable is also a source variable. Source variables are created by ActiveData for Oracle so that postbind and/or fetch data from one portion of a script can be used as input to subsequent bind statements.

 **Note:** If you are working with Oracle 8 select output data, use DO_OCI8GetSelectData instead.

If the formatType is INT_FORMAT, then the data will be converted to an integer before formatting (using atol()). This implies that the formatString will contain a %i, %d or equivalent.

Syntax

```
DO_GetSelectData( fetchCount, colnum, rowNum, srcName, formatType, formatString,
addConstant );
```

Parameters

Parameter	Description
fetchCount	A number from 1-n indicating which fetch sequence to use to fetch the data. The script code for a fetch statement is generally output as a C-based while-loop. This loop will retrieve data until no more data is available. This parameter determines which iteration of that loop to use to retrieve the data.
colnum	Column number to use to fetch the data. The first column is 1.
rowNum	Row number to use to fetch the data. The first row number is 1.
srcName	Pointer to the address of a source variable. The function will allocate memory for the source value and copy its value into this variable. Note

	that this parameter is a char **.
formatType	Data type to be used in the special format string. Acceptable values are: _NONE_, No special formatting. INT_FORMAT, Convert bind data to an integer before formatting. STRING_FORMAT, Assume that the bind data is numeric.
formatString	A printf-style format. The data will be formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the fetch data before conversion.

Example

The following example copies the fetched value of the first select-list item of the second row (in the first fetch iteration which retrieves 409 rows) to program variable FD_stmtnt_3_col_1_row_2.

It also copies the fetched value of the fifth select-list item of the second row (in the first fetch iteration) to program variable FD_stmtnt_3_col_5_row_2.

```
DO_oexec( CDA(1) ); /* Exec for statement 3 */
while ( DO_process_select_list( CDA(1), 409 ) )
{
DO_GetSelectData( FETCH(1), COL(1), ROW(2), &FD_stmtnt_3_col_1_row_2, _NONE_, "", 0 );
DO_GetSelectData( FETCH(1), COL(5), ROW(2), &FD_stmtnt_3_col_5_row_2, _NONE_, "", 0 );
} /* end of DO_process_select_list */
```

DO_init_alen

A routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_ScalarBindA.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function is not always called; for example, a script may not contain any DO_ScalarBindA calls, or the bind calls that are contained in the script do not utilize pAlen. This function should not be moved or modified.

Syntax

```
DO_init_alen(make_alen,pAlen,ALEN_COUNT);
```

Parameters

Parameter	Description
make_alen	A pointer to an array that holds the values of the length of data.
pAlen	A pointer to make_alen. Each element holds the pointer to the corresponding make_alen. In the example pAlen[0] = &make_alen[0], the contents of make_alen[0] are assigned before the call to DO_ScalarBindA.
ALEN_COUNT	The number of pAlen utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define ALEN_COUNT at the beginning of the script. Every bind does not necessarily utilize a pAlen. For example, if the alen was captured as NULL, NULL replaces the use of pAlen.

Example

```
#define ALEN_COUNT 20
:
sb2* pIndp[ALEN_COUNT]; /* sb2 is a signed integer */
sb2 make_alen[ALEN_COUNT];
:
DO_init_alen( make_alen, pAlen, ALEN_COUNT );
:
make_indp[1]=0;
make_alen[0]=90;
DO_ScalarBindA( CDA(3), ":id", _STRING, -1, pAlen[0], pIndp[1],"id", STRING_3_id_69 );
```

DO_init_data

A routine that allocates and initializes all of the logon data areas, cursor data areas, structures, and so on which are used to run the script.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. It should not be modified or removed.

Syntax

```
DO_init_data(s_info, LOGON_COUNT, CURSOR_COUNT, HANDLE_COUNT, DESCRIPTOR_COUNT);
```

Parameters

Parameter	Description
s_info	Structure used by each virtual user.
LOGON_COUNT	The number of logons in the script. If this number is incorrect, the script will fail. The number can be modified in the #define LOGON_COUNT at the beginning of the script.
CURSOR_COUNT	The number of cursors opened in the script. If this number is incorrect, the script will fail. The number can be modified in the #define CURSOR_COUNT at the beginning of the script.
HANDLE_COUNT	The number of handles opened in the script. If this number is incorrect, the script will fail. The number can be modified in the #define HANDLE_COUNT at the beginning of the script.
DESCRIPTOR_COUNT	The number of descriptors opened in the script. If this number is incorrect, the script will fail. The number can be modified in the #define DESCRIPTOR_COUNT at the beginning of the script.

Example

```
#define LOGON_COUNT 5
#define CURSOR_COUNT 35
#define HANDLE_COUNT 9
#define DESCRIPTOR_COUNT 1
:
:
DO_init_data( s_info, LOGON_COUNT, CURSOR_COUNT, HANDLE_COUNT, DESCRIPTOR_COUNT);
```

DO_init_indp

A routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_Bindv and DO_ScalarBindA calls.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function should not be moved or modified.

Syntax

```
DO_init_indp(make_indp,pIndp, INDP_COUNT);
```

Parameters

Parameter	Description
make_indp	A pointer to an integer array that holds the values of the null indicator variable.
pIndp	A pointer to make_indp. Each element holds the pointer to the corresponding null indicator variable. In the example pIndp[0] = &make_indp[0], the contents of make_indp[0] is assigned before the call to DO_BindV or DO_ScalarBindA.
INDP_COUNT	The number of indicator variables utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define INDP_COUNT at the beginning of the script. Every bind does not necessarily utilize a pIndp. If the null indicator was captured as NULL, NULL replaces the use of pIndp.

Example

```
#define INDP_COUNT 20
sb2* pIndp[INDP_COUNT]; /* sb2 is a signed integer */
sb2 make_indp[INDP_COUNT];
DO_init_indp(make_indp,pIndp, INDP_COUNT);
```

DO_oclose

Disconnects a previously opened cursor, returning all resources back to the Oracle server.

Syntax

```
DO_oclose( cursor );
```

Parameters

Parameter	Description
cursor	Cursor table index.

Equivalent OCI

```
oclose
```

Example

```
DO_oclose( CDA(0) );
```

DO_oexec

Executes the SQL statement associated with a cursor.

Before calling DO_oexec, the SQL statement must be parsed by calling DO_Oparse using the same cursor.

Syntax

```
DO_oexec( cursor );
```

Parameters

Parameter	Description
cursor	Cursor table index.

Equivalent OCI

oexec

Example

This example shows parsing and execution of a SQL statement:

```
DO_oparse( CDA(1), "select id, coname from company" );
DO_oexec( CDA(1) );
```

DO_olog

Establishes a connection between QALoad and an Oracle database.

An application must log in to Oracle before it can perform any other operations. Multiple connections to one or more Oracle instances is supported.

A user ID string is made up of the user's login ID, password, and an Oracle connection string.

A forward slash (/) separates the password from the user ID, and the connection string is preceded by the @symbol.

Syntax

```
DO_olog( LDAIndex, connect-string );
```

Parameters

Parameter	Description
LDAIndex	Logon data area index.
connect-string	A null terminated string containing the Oracle login ID, password, and connection.

Equivalent OCI

orlon

Example

This example shows a typical Oracle login sequence:

```
DO_olog( LDA(0), "scott/tiger@domain" );
```

DO_ologof

Closes a connection to the Oracle server, freeing its resources.

Syntax

```
DO_ologof( LDAIndex );
```

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

ologof

Example

```
DO_ologof( LDA(0) );
```

DO_oopen

Opens a cursor to the database.

Processing commands such as DO_oparse and DO_oexec require an open cursor. There may be multiple open cursors at one time, so operations may be repeated without re-parsing the SQL statement. QALoad automatically manages cursor opens and closes.

Syntax

```
DO_oopen( IdaIndex, cursor );
```

Parameters

Parameter	Description
IdaIndex	Logon data area index.
cursor	Cursor table index (first index is 0).

Equivalent OCI

oopen

Example

This example shows how a cursor is opened, a command is executed, and the cursor is subsequently closed:

```
DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1), "select id, coname from company" );
DO_oexec( CDA(1) );
DO_process_select_list( CDA(1), 100 ); /* get 100 rows */
DO_oclose( CDA(1) );
```

DO_oopt

Sets rollback options for non-fatal errors on multi-row INSERT and UPDATE SQL statements and determines whether to wait for requested resources or return errors.

Syntax

```
DO_oopt( cursor, rbopt, waitopt );
```

Parameters

Parameter	Description
cursor	Cursor table index.
rbopt	0 = Rollback on any error. 2 = Rollback only the failing row.
waitopt	0 = Wait indefinitely for resources to be available. 4 = Return an error if a resource is requested, but not available.

Equivalent OCI

oopt

DO_oparse

Parses a SQL statement or a PL/SQL block and associates it with a cursor index.

QALoad scripts use deferred mode linking and DO_oparse will defer the parse. In this mode, SQL statements are not actually sent to the server until the DO_oexec call. Therefore, SQL syntax errors are not reported at DO_oparse, but rather at DO_oexec.

Syntax

```
DO_oparse( cursor, statement );
```

Parameters

Parameter	Description
cursor	Cursor table index.
statement	Pointer to null terminated string containing the SQL statement.

Equivalent OCI

oparse

Example

This example shows a complete parse, execute, and fetch cycle for a SQL Select statement:

```
DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1), "select id, coname from company" );
DO_oexec( CDA(1) );
DO_process_select_list( CDA(1), 100 ); /* get 100 rows */
DO_oclose( CDA(1) );
```

DO_process_select_list

Fetches select-list data from the Oracle database. It is generally called repeatedly until there are no more rows satisfying the SQL select request.

The first time `DO_process_select_list` retrieves data for a SQL statement, it loops through all the returned fields (using `odescr`) and builds up a set of internal buffers to store the returned data. All data is returned as ASCII strings.

Syntax

```
DO_process_select_list( cursor, rowcount );
```

Parameters

Parameter	Description
cursor	Cursor table index.
rowcount	Number of rows to fetch into the buffer.

Equivalent OCI

`ofen`, `odescr`, and `odefin`

Example

This example shows the `DO_process_select_list` being called repeatedly, so all the rows are read:

```
DO_oexec( CDA(0) ); /* Exec for statement 2 */
while ( DO_process_select_list( CDA(0), 30 ) ); /* Read all rows, 30 at a time. */
```

DO_rollback

Rolls back the current transaction.

Syntax

```
DO_rollback( ldaIndex );
```

Parameters

Parameter	Description
ldaIndex	Logon data area index.

Equivalent OCI

`orol`

Example

```
DO_rollback( LDA(0) );
```

DO_ScalarBindA

Binds a program variable to a bind variable in a SQL statement.

A `DO_ScalarBindA` is generated wherever an `obndra` occurred in the capture file and the type of bind was a single (scalar) value, not an array. `DO_ScalarBindA` accurately reproduces the original bind call made by the application.

This eliminates extra data conversion steps and improves handling of OUTPUT variables to Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:).

DO_ScalarBindA must be called after the DO_oparse and before a DO_oexec. Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable.

Currently, DO_ScalarBindA does not support packed decimal, PCC-descriptor, cursor, mslable, oslabel, and any new Oracle 8 datatypes.

Syntax

```
DO_ScalarBindA (index, name, type, progvl, alen, indp, input, progv);
```

Parameters

Parameter	Description
index	Cursor table index.
name	Pointer to name of the bind variable (null terminated).
type	External datatype of bind variable.
progvl	Size of progv. This is the maximum size of the buffer. If binding an OUTPUT variable, progvl must be at least as large as the expected output value.
alen	Pointer to variable representing length of data. palen[0] points to the value of make_alen[0].
indp	Pointer to a null indicator variable. plndp[0] points to the value of make_indp[0]; If make_indp=SET_NULL, null will be passed as the data for the input. Otherwise, the data is passed as shown in the bind call.
input	Pointer to buffer containing input data.
progv	Output data buffer.

Equivalent OCI

obndra

Example

This example shows a Select command with two bind variables, :empid and :id.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID = :empid AND EMP_DEPT_ID = :id" );
make_indp[0]=0
DO_ScalarBindA(CDA(3), ":empid", _STRING, -1, NULL, pIndp[0], "200", STRING_3_empid_69);
make_indp[1]=0
make_alen[0]=90;
DO_ScalarBindA(CDA(3), ":id", _STRING, -1, pAlen[0], pIndp[1], "id", STRING_3_id_69);
DO_oexec( CDA(0) );
```

DO_SoftClose

Closes a cursor without destroying its resources on the server.

If the application is in the deferred mode, the cursor is not actually closed but is placed on a cursor-free list on the client. During any subsequent open cursor calls, the free list is checked first to satisfy the request. A

soft close reduces communication with the server because it does not cancel the cursor. An SQL statement associated with the cursor remains valid until the cursor is reused to pause another SQL statement.

Syntax

```
DO_SoftClose( cursor );
```

Parameters

Parameter	Description
cursor	Cursor date area index.

Example

This example shows how to use a soft close to execute a SQL statement again, then to parse and execute another SQL statement.

```
DO_oopen( LDA(0), CDA(0) );
DO_oparse( CDA(0), "select * from emp");
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 ) );

/* Soft close cursor */
DO_SoftClose( CDA(0) );

/* Reuse cursor to repeat statement */
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 ) );

/* Reuse cursor to parse next statement */
DO_oparse( CDA(0), "select ename from emp where empno = 7788");
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 ) );
DO_oclose( CDA(0) );
```

Oracle 7/8

Oracle 7/8 Index

DO_free_data

A required cleanup routine inserted into a script when it is generated and called before exiting the script. It should not be modified or moved.

DO_freitem

Frees the memory associated with an ActiveData for Oracle variable. The ActiveData variable (source variable) may have been assigned by DO_GetSelectData, DO_OC18GetSelectData, DO_strdup or DO_GetOutputData.

DO_GetOutputData

Copies the data from a bind variable into a source variable. Use with both Oracle 7 and 8 binds.

DO_makedate

Binds dates into a SQL statement in C-based scripts

DO_strdup

Places the data retrieved by ActiveData for Oracle from a QALoad central datapool or a local datapool.

DO_free_data

A required cleanup routine inserted into a script when it is generated and called before exiting the script. It should not be modified or moved.

Syntax

```
DO_free_data( );
```

Parameters

None.

DO_freeitem

Frees the memory associated with an ActiveData for Oracle variable.

The ActiveData variable (source variable) may have been assigned by DO_GetSelectData, DO_OC18GetSelectData, DO_strdup or DO_GetOutputData.

The memory assigned for variables by ActiveData for Oracle is allocated from the program's free memory area (malloc). This function releases that memory. It is automatically placed in your script for all variables created by the Oracle conversion program.

If you add your own variables, you should include a DO_freeitem() to release allocated memory at the end of the transaction loop.

Syntax

```
DO_freeitem( name );
```

Parameters

Parameter	Description
name	Pointer to source variable.

DO_GetOutputData

Copies the data from a bind variable into a source variable. Use with both Oracle 7 and 8 binds.

Source variables are created by ActiveData for Oracle so that postbind or fetch data from one portion of a script can be used as input to subsequent bind statements.

DO_GetOutputData() allocates memory for the contents of the source from the system's free memory pool (malloc).

If the formatType is INT_FORMAT, then the data is converted to an integer before formatting (using atoi()). This implies that the formatString contains a %i, %d or equivalent.

Syntax

```
DO_GetOutputData( srcName, type, bindName, length, formatType, formatString, addConstant );
```

Parameters

Parameter	Description
srcName	Pointer to the address of a source variable. The function allocates memory for the source value, and copies its value into this variable. Note that this

	parameter is a char **.
type	External datatype of the bindName. Presently, only character and numeric data types are supported. Binary dates and rowids are not.
bindName	Pointer to a variable that contains the bind data to be copied.
length	Length of the bind data.
formatType	Data type to be used in the special format string. Acceptable values are: _NONE_, No special formatting. INT_FORMAT, Convert bind data to an integer before formatting. STRING_FORMAT, Assume that the bind data is numeric.
formatString	A printf-style format. The data is formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the bindName before conversion.

Example

```
DO_OCISmtExecute( HNDL(6) ); /* Exec for statement 6 */
DO_GetOutputData(&PB_XHOME_TELE, _FIXED_CHAR,
    CHAR_2_XHOME_TELE_9, *pAlen[65],
    _NONE_, "", 0);
DO_GetOutputData(&PB_NEXT_ID, _FIXED_CHAR,
    CHAR_2_ID, *pAlen[65], INT_FORMAT,
    "%04i", ADD(1) );
```

DO_makedate

Binds dates into a SQL statement in C-based scripts

Since standard date formats differ between countries, QALoad implements a language/ country independent method of binding dates into a SQL statement. When DO_makedate is used in conjunction with a DO_BindV, dates are properly processed regardless of the currently selected date format. If QALoad detects a bind variable representing a date, it automatically declares a variable of type ORADATE and generates the appropriate DO_makedate call.

Syntax

```
DO_makedate( date_var, year, month, day, hour, min, second );
```

Parameters

Parameter	Description
date_var	Pointer to a variable of type ORADATE.
year	Four-digit year.
month	Two-digit month.
day	Two-digit day of the month.
hour	Hour in the day (0 to 23).
min	Minute within the hour (0-59).

second	Seconds within the minute (0-59).
--------	-----------------------------------

Example

This example shows how to get a date ready for use in a SQL statement:

```
ORADATE DATE_0_8;
...
DO_makedate( &DATE_0_8, 1995, 7, 12, 0, 0, 0 );
DO_BindV (CDA(0); "(text*):8, _DATE,7,NULL, (ub1*) &DATE_0_8, (ub1*) &DATE_0_8);
```

DO_strdup

Places the data retrieved by ActiveData for Oracle from a QALoad central datapool or a local datapool.

ActiveData for Oracle enables you to read data from a central datapool using the GET_DATA() script command, and data from a local datapool using the READ_DATA_RECORD() script command. The DO_strdup script command places a specific datapool field value in a program variable. Typically, the program variable is also a source variable. Source variables are created by ActiveData for Oracle so that postbind and/or fetch data from one portion of a script can be used as input to subsequent bind statements.

Syntax

When using a local datapool:

```
DO_strdup( progVar, GET_DATA_FIELD( datapool_nbr, field_nbr ) );
```

When using a central datapool:

```
DO_strdup( progVar, VARDATA( field_nbr ) );
```

Parameters

Parameter	Description
progVar	Pointer to the address of a source variable. The function allocates memory for the source value and places the value of the second parameter into this variable. Note that this variable is a char**.
Datapool_nbr	Number assigned to the local datapool used by ActiveData for Oracle. The number is preset to 1.
Field_nbr	The field number in the datapool that serves as input. The number is internally set to 1.

Example

When using a central datapool:

```
char* Data_pkey = NULL;
:
GET_DATA();
:
DO_strdup( &Data_pkey, VARDATA(1) );
:
DO_olog( LDA(0), "scott/tiger@os816qal.world" );
```

When using a local datapool:

```
char* Data_pkey = NULL;
:
```

```

OPEN_DATA_POOL( "E:\\Program Files\\Compuware\\QALOAD\\
Middlewares\\Oracle\\Scripts\\LocalDataPool.DAT", ORA_DATAPool, TRUE );
:
BEGIN_TRANSACTION();
:
READ_DATA_RECORD( ORA_DATAPool );
:
DO_strdup( &Data_pkey, GET_DATA_FIELD( ORA_DATAPool, 1 ) );
:
DO_olog( LDA(0), "scott/tiger@os816qal.world" );

```

Oracle 8

Oracle 8 Index

[DO_OCI8BindDate](#)

Binds a date variable (created by DO_makedate) to a bind variable in a SQL statement.

[DO_OCI8BindNull](#)

Binds a NULL value to a bind variable in a SQL statement.

[DO_OCI8BindString](#)

Binds a program variable to a bind variable in a SQL statement.

[DO_OCI8GetSelectData](#)

Copies the data retrieved from an Oracle8 SQL SELECT statement into a program variable.

[DO_OCI8InitAlen](#)

An Oracle8-specific routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_OCIBind.

[DO_OCI8InitIndp](#)

An OCI8-specific routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_OCIBind.

[DO_OCIAttrSet](#)

Sets a particular attribute for a previously allocated Oracle 8 OCI handle.

[DO_OCIBind](#)

Binds a program variable to a bind variable in a SQL statement.

[DO_OCICommit](#)

Commits the current Oracle8 transaction. A commit should be performed after all relevant SQL statements have been processed.

[DO_OCIDefine](#)

Associates an item in a select-list to an Oracle external datatype and an output data buffer.

[DO_OCIDescriptorAlloc](#)

Allocates and initializes an Oracle 8 OCI descriptor or LOB locator.

[DO_OCIDescriptorFree](#)

De-allocates an Oracle 8 OCI descriptor or LOB locator.

[DO_OCIEnvFreeAll](#)

De-allocates all environment handles before the end of an OCI8 script.

[DO_OCIEnvInit](#)

Allocates and initializes an Oracle OCI8 environment handle.

[DO_OCIExecute](#)

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCISmtPrepare. Note that SQL syntax errors are reported at execution time.

[DO_OCISvcCtxAlloc](#)

Allocates and initializes an Oracle 8 OCI handle.

[DO_OCISvcCtxFree](#)

De-allocates an Oracle 8 OCI handle.

[DO_OCISvcCtxInitialize](#)

Initializes the Oracle OCI8 process environment. This command must be issued once in a QALoad script prior to any other Oracle8 script commands, and should be outside any QALoad transactions.

[DO_OCISvcCtxToLda](#)

Toggles an Oracle 7 logon data area to an Oracle 8 service context handle. This should be done after using DO_OCISvcCtxToLda to create Oracle 7 in a database session in Oracle 8.

[DO_OCISvcCtxRead](#)

Reads a LOB into a buffer.

[DO_OCISvcCtxWrite](#)

Writes the contents of a buffer into an Oracle 8 LOB.

[DO_OCISvcCtxLogoff](#)

Terminates an Oracle OCI8 logon session and connection created with DO_OCISvcCtxLogon.

[DO_OCISvcCtxLogoffEx](#)

Terminates an Oracle OCI8 logon session and connection created with DO_OCISvcCtxLogon.

[DO_OCISvcCtxLogon](#)

Creates a simple Oracle OCI8 logon connection and session for QALoad . Any application must log on to Oracle before performing any other Oracle operations.

[DO_OCISvcCtxProcessSelectList](#)

Fetches select-list data from an Oracle 8 database after an OCISvcCtxExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

[DO_OCISvcCtxProcessSelectList_EX](#)

Fetches select-list data from an Oracle 8 database after an OCISvcCtxExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

[DO_OCISvcCtxRollback](#)

Rolls back the current Oracle8 transaction.

[DO_OCISvcCtxServerAttach](#)

Creates a standard Oracle OCI8 database connection for QALoad . Note that individual Oracle 8 user logons are done with the DO_OCISvcCtxServerAttach command.

[DO_OCISvcCtxServerDetach](#)

Detaches QALoad from the Oracle OCI8 data source connection previously attached to with the DO_OCIServerAttach command. Note that all users must be logged off with the DO_OCISessionEnd command before this call.

DO_OCISessionBegin

Creates an Oracle OCI8 logon session for QALoad to a server previously attached to with DO_OCIServerAttach. Any application must log on to Oracle before performing any other Oracle operations.

DO_OCISessionEnd

Terminates an Oracle user session previously created with the DO_OCISessionBegin command.

DO_OCISmtExecute

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCISmtPrepare. Note that SQL syntax errors are reported at execution time.

DO_OCISmtPrepare

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

DO_OCISmtPrepare_EX

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

DO_OCISvcCtxToLda

Toggles an Oracle 8 service context handle to an Oracle 7 logon data area. This allows Oracle 7 cursors to be created in a database session created in Oracle 8.

DO_OCITransCommit

Commits the current Oracle 8 transaction. A commit should be performed after all relevant SQL statements have been processed.

DO_OCITransRollback

Rolls back the current Oracle 8 transaction.

Logging on and off Oracle Net 8

Command sequence for logging in to Oracle 8

```

/* NOTE: HNDL(0) is the environment handle. It should be previously specified in a
DO_OCIEnvInit call */

/* An error handle is used for Oracle8 error handling. HNDL(1) is the index to the new
error(OCI_HTYPE_ERROR) handle. */
DO_OCISetHandleAlloc( HNDL(0), HNDL(1), OCI_HTYPE_ERROR);

/* A server handle is allocated for DO_OCIServerAttach. HNDL(2) is the index to the new
server (OCI_HTYPE_SERVER) handle. */
DO_OCISetHandleAlloc( HNDL(0), HNDL(2), OCI_HTYPE_SERVER);

/* The DO_OCIServerAttach handle uses the server (HNDL(2)) handle previously allocated in
DO_OCISetHandleAlloc. Note that the TNS data source name is the third parameter in this call.
*/
DO_OCIServerAttach( HNDL(2), HNDL(1), "oracledb.world", 15, OCI_DEFAULT);

/* A service context handle is now allocated. HNDL(3) is the index to the new service
context (OCI_HTYPE_SVCCTX) handle. */
DO_OCISetHandleAlloc( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);

/* The allocated service context handle (HNDL(3)) is now set as an attribute of the server
handle(HNDL(2)) in this DO_OCISetAttr call. Note that the error handle (HNDL(1)) is a

```

QALoad 5.02

```
parameter in a DO_OCISessionBegin call. */
DO_OCISessionBegin( HNDL(3), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(1), HNDL(2));

/* A session handle is allocated for DO_OCISessionBegin. HNDL(3) is the index to the new
session (OCI_HTYPE_SESSION) handle. */
DO_OCISessionBegin( HNDL(0), HNDL(4), OCI_HTYPE_SESSION);

/* The username is set as an attribute of the session handle (HNDL(4)). */
DO_OCISessionBegin( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),
IS_ATTRIBUTE);

/* The password is set as an attribute of the session handle (HNDL(4)). */
DO_OCISessionBegin( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),
IS_ATTRIBUTE);

/* The DO_OCISessionBegin call uses the service context handle (HNDL(3)) and the session
handle (HNDL(4)). . Note that the error handle (HNDL(1)) is a parameter in a
DO_OCISessionBegin call. The Credentials parameter is OCI_CRED_RDBMS, which means that
username and password must have been explicitly set in previous calls to DO_OCISessionBegin. If
the user verification is integrated with external credentials, use OCI_CRED_EXT as this
value. When you use OCI_CRED_EXT, you will not have to set the username and password in
DO_OCISessionBegin calls prior to DO_OCISessionBegin. */
DO_OCISessionBegin( HNDL(3), HNDL(1), HNDL(4), OCI_CRED_RDBMS, OCI_DEFAULT);
```

Command sequence for logging off Oracle 8

```
/* The DO_OCISessionEnd call uses the same service context handle (HNDL(3)) and session
handle (HNDL(4)) used in DO_OCISessionBegin. Note that the error handle (HNDL(1)) is a
parameter in a DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(3), HNDL(1), HNDL(4), OCI_DEFAULT);

/* The session handle (HNDL(4)) is no longer needed, so it is de-allocated with the
DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(4), OCI_HTYPE_SESSION);

/* The DO_OCIServerDetach call uses the same server handle (HNDL(2)) and service context
handle (HNDL(1)) used in the DO_OCIServerAttach call. Note that the error handle (HNDL(1))
is a parameter in a DO_OCIServerDetach call. */
DO_OCIServerDetach( HNDL(2), HNDL(1), OCI_DEFAULT);

/* The server handle (HNDL(2)) is no longer needed, so it is de-allocated with the
DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(2), OCI_HTYPE_SERVER);

/* The service context handle (HNDL(3)) is no longer needed, so it is de-allocated with the
DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(3), OCI_HTYPE_SVCCTX);

/* If the error handle (HNDL(1)) is no longer needed, it should be de-allocated with the
DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(1), OCI_HTYPE_ERROR);
```

Using QALoad script commands to log on and off an Oracle8 database

1. Create the appropriate handles and increment the HANDLE_COUNT parameter in the QALoad script.
 - a. Find the number after HANDLE_COUNT in the QALoad script. Note the current number, and then add four (4) to this number. Four is the count of the number of new handles we will be allocating for use by this logon and logoff example.
For example if the following line is in the script:
HANDLE_COUNT 35
edit it to read:
HANDLE_COUNT 39
 - b. Then, allocate a server handle, a service context handle, a session handle and an error handle (or you may use another pre-allocated error handle; this error handle must not be freed before all logoff commands are called).
 - c. Associate the numbers 35, 36, 37, and 38 (starting with the previous HANDLE_COUNT and adding 1 for each new handle) with these new handles to be allocated, as shown in the following example:


```
DO_OCIServerAttach( HNDL(36), OCI_HTYPE_SERVER);
DO_OCIServerAttach( HNDL(37), OCI_HTYPE_SVCCTX);
DO_OCIServerAttach( HNDL(38), OCI_HTYPE_SESSION);
DO_OCIServerAttach( HNDL(35), OCI_HTYPE_ERROR);
```

2. Add the call to attach to the Oracle database (DO_OCIServerAttach). The following examples shows the code for the DO_OCIServerAttach call that uses the handles allocated in the previous step:

```
DO_OCIServerAttach( HNDL(36), HNDL(35), "oracledb.world", 15, OCI_DEFAULT);
```

3. Set the allocated service context handle as an attribute to the server handle with calls to DO_OCIServerAttach. Setting the service context handle as an attribute of the server handle allows the service context handle to be used in the DO_OCISessionBegin call. Ensure the handle indexes are correct, and keep the other parameters the same as shown below:

```
DO_OCIServerAttach( HNDL(37), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(35), HNDL(36));
```

4. If you are using Oracle security, set the session handle attributes. You will need to specify the username and password. Using the DO_OCIServerAttach calls will tie the username and password as attributes to the session handle. Ensure that the UserName and UserNameLength parameters are set correctly for this first DO_OCIServerAttach call, and the Password and PasswordLength attributes are set correctly for the second DO_OCIServerAttach call, as shown below:

```
DO_OCIServerAttach( HNDL(38), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(35), IS_ATTRIBUTE);
```

```
DO_OCIServerAttach( HNDL(38), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(35), IS_ATTRIBUTE);
```

5. Start the Oracle session with a call to DO_OCISessionBegin, as shown below. Note that the OCI_CRED_RDBMS parameter implies that the username and password are set with DO_OCIServerAttach calls, if using integrated security, step 4 is not needed, and use OCI_CRED_EXT as the parameter:

```
DO_OCISessionBegin( HNDL(37), HNDL(35), HNDL(38), OCI_CRED_RDBMS, OCI_DEFAULT);
```

6. After all SQL statements for the session have completed, log off the database as follows:

a. End the session with the DO_OCISessionEnd call:

```
DO_OCISessionEnd( HNDL(37), HNDL(35), HNDL(38), OCI_DEFAULT);
```

b. Disconnect from the server with the DO_OCIServerDetach call:

```
DO_OCIServerDetach( HNDL(36), HNDL(35), OCI_DEFAULT);
```

c. Free the allocated handles using DO_OCIServerDetach, or a memory leak will develop in the application:

```
DO_OCIServerDetach( HNDL(38), OCI_HTYPE_SESSION);
```

```
DO_OCIServerDetach( HNDL(36), OCI_HTYPE_SERVER);
```

```
DO_OCIServerDetach( HNDL(37), OCI_HTYPE_SVCCTX);
```

```
DO_OCIServerDetach( HNDL(35), OCI_HTYPE_ERROR);
```

Oracle SQL statements in Oracle 8 with QALoad script commands

SQL Statement Types

QALoad scripts support the following standard SQL statements:

! SQL data manipulation language (DML) statements

Examples of DML statements include:

```
QSELECT * FROM USER_TAB;
```

```
INSERT INTO EMP (VALUES "John Doe", "Accounting", 20, 500.00);
```

```
DELETE FROM DEPT WHERE DEPTNO = 100;
```

```
UPDATE EMP SET DEPTNO = 40 WHERE DEPTNO = 50;
```

! Anonymous PL/SQL blocks

An example of anonymous PL/SQL blocks includes:

```
BEGIN UPDATE EMP SET PAY = 400.00; END;
```

! SQL data definition language (DDL) statements

An example of DDL statements includes:

```
CREATE TABLE emp (empno NUMBER(5) PRIMARY KEY);
```

! PL/SQL stored procedure or function calls

An example of a PL/SQL stored procedure call includes:

```
"BEGIN qaload_regtest.emptytest(:pkey, :f1, " ":num2, :opkey, :of1, :onum2); END;
```

Note that QALoad does not support Oracle 8 objects, Oracle 8 user-defined types (UDTs) or Oracle 8 reference pointers.

Command sequence to read a LOB into a memory buffer

Following is a sample code sequence to read a 1024-byte LOB from a memory buffer into an Oracle 8 LOB parameter as part of an INSERT statement. Note that QALoad will create a temporary memory buffer and will populate it with meaningless data. Comments are added to commands where appropriate.

```
DO_OCIBHandleAlloc( HNDL(0), HNDL(5), OCI_HTYPE_STMT);

/* A special descriptor (often referred to as a lob locator) must be created for the lob
object. Note that the DESCRIPTOR_COUNT value in the script will need to be incremented by 1
and that the 2nd parameter in the call to DO_OCIDescriptorAlloc is DESC(n) where n is the
previous value in DESCRIPTOR_COUNT */
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);

/* DO_OCISstmtPrepare( HNDL(5), "INSERT INTO CLBTAB VALUES ( 'Test', " "EMPTY_CLOB())",
OCI_NTV_SYNTAX );

/* Note that since the LOB is empty, there is no bind call before the execute call. */
DO_OCIBExecute( HNDL(5), 1, OCI_DEFAULT );

/* Use the DO_OCIBlobWrite call to write the LOB data. Note that 1024 bytes are being written
to the LOB column of the inserted record. */
DO_OCIBlobWrite(HNDL(3), HNDL(1), DESC(0), 1024, 1, 1024, 0, 0, 1 );

/* Once the LOB is written, the descriptor is freed with a call to DO_OCIDescriptorAlloc */
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
DO_OCIBHandleFree( HNDL(5), OCI_HTYPE_STMT);
```

Command sequence to write a LOB from a memory buffer

Below is a sample code sequence to write a 1024-byte LOB from an Oracle 8 LOB to a memory buffer. Note that QALoad will create a temporary memory buffer to store the data. Comments are added to commands where appropriate.

```
DO_OCIBHandleAlloc( HNDL(0), HNDL(6), OCI_HTYPE_STMT);

/* A special descriptor (often referred to as a lob locator) must be created for the lob
object. Note that the DESCRIPTOR_COUNT value in the script will need to be incremented by 1
and that the 2nd parameter in the call to DO_OCIDescriptorAlloc is DESC(n) where n is
the previous value in DESCRIPTOR_COUNT */
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCISstmtPrepare( HNDL(6), "SELECT essay FROM CLBTAB WHERE name = 'Test' " "for update",
OCI_NTV_SYNTAX );

/* Since the LOB is an output from the SELECT statement, a DO_OCIBDefine call must associate
the select parameter with the allocated descriptor. See DO_OCIBDefine for more information.
*/
DO_OCIBDefine(HNDL(6), HNDL(1), 1, 1, SQLT_CLOB, 0, DESC(0));
DO_OCIBExecute( HNDL(6), 1, OCI_DEFAULT );

/* Use the DO_OCIBlobRead call to read the LOB data from the database. Note that 1024 bytes
are being read from the LOB column of the fetched record. */
DO_OCIBlobRead(HNDL(3), HNDL(1), DESC(0), 1000, 1, 1024, 0, 1 );

/* Once the LOB is read, the descriptor is freed with a call to DO_OCIDescriptorAlloc */
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
DO_OCIBHandleFree( HNDL(6), OCI_HTYPE_STMT);
```

DO_OCIBindDate

Binds a date variable (created by DO_makedate) to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:).

DO_OCIBindDate commands must be placed between the DO_OCISmtPrepare command and the DO_OCISmtExecute command. To bind by position, instead of by name, precede the position bind variable with an "@" symbol.

DO_OCIBindDate is a deprecated command. It is recommended that you use DO_OCIBind instead.

Syntax

```
DO_OCIBindDate( statementHandleIndex, BindVariableName, &ORADATEStructPtr );
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCISmtPrepare.
BindVariableName	The name of the bind variable as a character string.
&ORADATEStructPtr	A pointer to an ORADATE structure, define in Oracle's header files.

Equivalent OCI

OCIBindByName, OCIBindByPos

Example

The following example shows the fetch loop to retrieve data after a SQL statement is executed.

```
DO_OCIBindDate( HNDL(5), ":CURDATE", &CURDATE1);
```

DO_OCIBindNull

Binds a NULL value to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_OCIBindString commands must be placed between the DO_OCISmtPrepare command and the DO_OCISmtExecute command. To bind by position instead of by name, precede the position bind variable with an "@" symbol.

Syntax

```
DO_OCIBindString(statementHandleIndex, BindVariableName );
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCISmtPrepare.
BindVariableName	The name of the bind variable as a null-terminated character string.

Equivalent OCI

OCIBindByName, OCIBindByPos

Example

The following example shows the :DUMMY bind variable being bound to a NULL value.

```
DO_OCIBindNull(HNDL(5), ":DUMMY");
```

DO_OCIBindString

Binds a program variable to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_OCIBindString commands must be placed between the DO_OCISmtPrepare command and the DO_OCISmtExecute command. To bind by position instead of by name, precede the position bind variable with an "@" symbol.

DO_OCIBindString only supports the binding of strings, nulls, or dates. If you need to bind a numeric value, convert it first to a string before passing it to DO_OCIBindString. If needed, Oracle automatically converts character data types to numeric.

DO_OCIBindString is a deprecated command. It is recommended that you use DO_OCIBind instead.

DO_OCIBindString binds every data type as a fixed character and forces the Oracle server to make implicit database conversions. Also, you must variablize OUTPUT variables or they will overwrite the input data held by string constants.

Syntax

```
DO_OCIBindString(statementHandleIndex, BindVariableName, ValueString );
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCISmtPrepare.
BindVariableName	The name of the bind variable as a null-terminated character string.
ValueString	A pointer to a string containing the value for the bind variable (null terminated).

Equivalent OCI

OCIBindByName, OCIBindByPos

Example

The following example shows the program variable CITYNAME being bound to the :CITY bind variable.

```
DO_OCIBindString( HNDL(5), ":CITY", CITYNAME );
```

DO_OCIGetSelectData

Copies the data retrieved from an Oracle8 SQL SELECT statement into a program variable.

DO_OCIGetSelectData processes the data retrieved by DO_OCIProcessSelectList() by copying the value of the fetched data to another program variable. Typically, the program variable is also a source variable. Source variables are created by ActiveData for Oracle so that postbind and/or fetch data from one portion of a script can be used as input to subsequent bind statements.

 **Note:** If you are working with Oracle 7 select output data, use DO_GetSelectData instead.

If the formatType is INT_FORMAT, then the data is converted to an integer before formatting (using atoi()).

This implies that the formatString contains a %i, %d or equivalent.

Syntax

```
DO_OCISGetSelectData( fetchCount, colnum, rowNum, srcName, formatType, formatString,
addConstant );
```

Parameters

Parameter	Description
fetchCount	A number from 1-n indicating which fetch sequence to use to fetch the data. The script code for a fetch statement is generally output as a C-based while-loop. This loop retrieves data until no more data is available. This parameter tells which iteration of that loop to use to retrieve the data.
colnum	Column number to use to fetch the data. The first column is 1.
rowNum	Row number to use to fetch the data. The first row number is 1.
srcName	Pointer to the address of a source variable. The function allocates memory for the source value and copy its value into this variable. Note that this parameter is a char**.
formatType	Data type to be used in the special format string. Acceptable values are: _NONE_: no special formatting INT_FORMAT: convert bind data to an integer before formatting STRING_FORMAT: assume that the bind data is numeric.
formatString	A printf-style format. The data is formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the fetch data before conversion.

Example

The following example copies the fetched value of the first select-list item of the second row (in the first fetch iteration which retrieves 409 rows) to program variable FD_stmnt_3_col_1_row_2. It also copies the fetched value of the fourth select-list item of the seventh row (in the first fetch iteration) to program variable FD_stmnt_3_col_4_row_7.

```
DO_OCISstmtExecute( HNDL(6)); /* Exec for statement 3 */
while (DO_OCIProcessSelectList (HNDL(6), 409) )
{
DO_OCISGetSelectData (FETCH(1), COL(1), ROW(2),
&FD_stmnt_3_col_1_row_2,
_NONE_, "", 0);
DO_OCISGetSelectData(FETCH(1), COL(4), ROW(7),
&FD_stmnt_3_col_4_row_7,
INT_FORMAT, "%04i", ADD(3));
DO_OCISGetSelectData (FETCH(1), COL(5), ROW(2),
&FD_stmnt_3_col_5_row_2,
_NONE_, "", 0);
} /* end of DO_OCIProcessSelectList */
```

DO_OCI8InitIndp

An OCI8-specific routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_OCIBind.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function should not be moved or modified.

Syntax

```
DO_OCI8InitIndp( makeOCI8Indp, pOCI8Indp, OCI8_INDP_COUNT );
```

Parameters

Parameter	Description
makeOCI8Indp	A pointer to an integer array that holds the values of the null indicator variable.
pOCI8Indp	A pointer to makeOCI8Indp. Each element holds the pointer to the corresponding null indicator variable. In the example pOCI8Indp [0] = & makeOCI8Indp[0], the contents of makeOCI8Indp[0] is assigned before the call to DO_OCIBind.
OCI8_INDP_COUNT	The number of indicator variables utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define OCI8_INDP_COUNT at the beginning of the script. Every bind does not necessarily utilize a pOCI8Indp. If the null indicator was recorded as NULL, NULL replaces the use of pOCI8Indp.

Example

```
#define OCI8_INDP_COUNT 20
:
:
sb2* pOCI8Indp [OCI8_INDP_COUNT]; /* sb2 is a signed integer
:
sb2 makeOCI8Indp [OCI8_INDP_COUNT];
:
:
DO_OCI8InitIndp( makeOCI8Indp, pOCI8Indp, OCI8_INDP_COUNT );
:
:
makeOCI8Indp[0]=0;
makeOCI8Alen[0]=4;
DO_OCIBind( HNDL(9), HNDL(2), "@2", _INTEGER, 4,
            pOCI8Alen[0], pOCI8Indp[0], (ub1 *) "0",
            (ub1 *) &INTEGER_9_2_0 );
```

DO_OCI8InitAlen

An Oracle 8-specific routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_OCIBind.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function is not always called. For example, a script may not contain any DO_OCIBind calls, or the bind calls that are contained in the script do not utilize pOCI8Alen. This function should not be moved or modified.

Syntax

```
DO_OCIBind( makeOCI8Alen, pOCI8Alen, OCI8_ALEN_COUNT );
```

Parameters

Parameter	Description
makeOCI8Alen	A pointer to an array that holds the values of the length of data.
pOCI8Alen	A pointer to makeOCI8Alen. Each element holds the pointer to the corresponding makeOCI8Alen. In the example, pOCI8Alen [0] = &makeOCI8Alen [0], the contents of makeOCI8Alen [0] are assigned before the call to DO_OCIBind.
OCI8_ALEN_COUNT	The number of pOCI8Alen utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define OCI8_ALEN_COUNT at the beginning of the script. Every bind does not necessarily utilize a pOCI8Alen. For example, if the alen was captured as NULL, NULL replaces the use of pOCI8Alen.

Example

```
#define OCI8_ALEN_COUNT 20
:
:
sb2*pOCI8Alen [OCI8_ALEN_COUNT]; /* sb2 is a signed integer */
sb2 makeOCI8Alen [OCI8_ALEN_COUNT];
:
:
DO_OCIBind( makeOCI8Alen, pOCI8Alen, OCI8_ALEN_COUNT );
:
:
makeOCI8Indp[0]=0;
makeOCI8Alen[0]= 4;
DO_OCIBind( HNDL(9), HNDL(2), "@2", _INTEGER,
            4, pOCI8Alen[0], pOCI8Indp[0],
            (ub1 *) "0", (ub1 *) &INTEGER_9_2_0 );
```

DO_OCIBind

Sets a particular attribute for a previously allocated Oracle 8 OCI handle.

Syntax

```
DO_OCIBind(targetHandleIndex, targetHandleType, attributep, attributeSize, attributeType,
errorHandleIndex);
```

Parameters

Parameter	Description
targetHandleIndex	An index to a previously allocated Oracle handle whose attribute is to be set.
targetHandleType	The handle type.
attributep	A pointer to the attribute value. This can be a character string or another handle in select instances.
attributeSize	The size of the attribute value.

attributeType	The type of attribute to set for the handle.
errorHandleIndex	An index to a previously allocated Oracle 8 error handle.

Equivalent OCI

OCIAttrSet

Example

This example sets OCI_ATTR_USERNAME and OCI_ATTR_PASSWORD attributes of the session handle prior to calling DO_OCISessionBegin to start an Oracle8 session on a previously attached database.

```
DO_OCISessionBegin( HNDL(5), OCI_HTYPE_SESSION, "scott", 5,
OCI_ATTR_USERNAME, HNDL(1), IS_ATTRIBUTE);
DO_OCISessionBegin( HNDL(5), OCI_HTYPE_SESSION, "tiger", 5,
OCI_ATTR_PASSWORD, HNDL(1), IS_ATTRIBUTE);
```

DO_OCIBind

Binds a program variable to a bind variable in an SQL statement.

A DO_OCIBind is generated wherever a bind occurs in the capture file. Note that the binds only support single (scalar) values, not array values. DO_OCIBind accurately reproduces the original bind call made by the application. This eliminates extra data conversion steps and improves handling of OUTPUT variables in Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:). DO_OCIBind must be called after the DO_OCISmtPrepare and before a DO_OCISmtExecute.

Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable. Currently, DO_OCIBind only supports the datatypes supported by DO_ScalarBindA.

Syntax

```
DO_OCIBind(statementHandleIndex, errorHandleIndex, BindVariable, DataType,
OutputBufferLength, OCI8Alen, OCI8Indp, InputBuffer, OutputBuffer);
```

Parameters

Parameter	Description
statementHandleIndex	An index to the current allocated Oracle 8 statement handle used in the DO_OCISmtPrepare call.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
BindVariable	A pointer to the name of the null-terminated bind variable string.
DataType	External datatype of the bind variable. Valid Oracle external datatypes (with program variable types) include: SQLT_CHR (char[n]) SQLT_NUM (unsigned char[21]) SQLT_INT (signed char) SQLT_FLT (float, double) SQLT_STR (char[n+1]) SQLT_VNU (char[22]) SQLT_LNG (char[n])

	SQLT_VCS (char[n] + sizeof(short int)) SQLT_DAT (char[7]) SQLT_VBI (unsigned char[n + sizeof(short int)]) SQLT_BIN (unsigned char[n]) SQLT_LBI (unsigned char[n]) SQLT_UIN (unsigned) SQLT_LVC (char[n + sizeof(int)]) SQLT_LVB (unsigned char[n + sizeof(int)]) SQLT_AFC (char[n]) SQLT_AVC (char[n + 1]) SQLT_CLOB see LOB example SQLT_BLOB see LOB example SQLT_FILE see LOB example SQLT_FILE see LOB example.
OutputBufferLength	Size of the output buffer. This is the maximum size of the OutputBuffer buffer. If binding a PL/ SQL OUTPUT variable, this value must be at least as large as the expected output variable.
OCI8Alen	Pointer to a variable that contains the length of the bind data. This is an alternative method of defining the length of the output data. Use the makeOCI8Alen macro to create this pointer. OCI8Alen should only be used if it is necessary to determine the length of the bind value returned from a statement execute. For character strings, using the strlen on the OutputBuffer variable after the statement execute is an easier method of obtaining this length.
OCI8Indp	Pointer to an indicator that the bind variable is NULL. Use the makeOCI8Indp macro to create this pointer. Using the DO_OCIBindNull call is a preferred way of binding a NULL value unless the bind variable is a PL/SQL OUTPUT variable.
InputBuffer	Pointer to a buffer containing the input data.
OutputBuffer	Pointer to the output data buffer.

Equivalent OCI

OCIBindByName, OCIBindByPos


Example

```
DO_OCIBind(HNDL(5), HNDL(1), ":PKEY", _VARCHAR2, strlen(PB_PKEY), NULL, NULL, (ub1 *)
PB_PKEY, (ub1 *) VARCHAR2_6_PKEY_1);
```

DO_OCICommit

Commits the current Oracle8 transaction. A commit should be performed after all relevant SQL statements have been processed.

DO_OCICommit is a deprecated command. It is recommended that you use DO_OCITransCommit instead.

 **Note:** DO_OCICommit should only be used in a single-user environment. For multi-user environments, use DO_OCITransCommit.

Syntax

```
DO_OCICommit (errorHandleIndex, CommitType);
```

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to commit. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for QALoad scripts.

Equivalent OCI

OCITransCommit

Example

The following example shows a SQL statement being committed after the execute and fetch loop.

```
DO_OCISstmtExecute( HNDL(5) ); /* Exec for statement 3 */
DO_OCICCommit( HNDL(5), HNDL(1), OCI_DEFAULT );
```

DO_OCIDefine

Associates an item in a select-list to an Oracle external datatype and an output data buffer.

Syntax

```
DO_OCIDefine(statementHandleIndex, errorHandleIndex, FetchCount, SelectListPosition,
DataType, BufferLength, lobDescriptorIndex);
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously used in the call to DO_OCISstmtPrepare.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
FetchCount	The fetch count as defined in DO_OCIPProcessSelectList. This value should be set to the value defined to DO_OCIPProcessSelectList or to 1.
selectListPosition	The position of the item in the select list. The starting point is 1.
DataType	The Oracle external datatype.
BufferLength	The maximum data length for the output buffer for the defined value.
lobDescriptorIndex	An index to an previously allocated Oracle 8 lob locator, if the output is a BLOB, CLOB, or BFILE. If a lob locator is not used, the value is IS_ATTRIBUTE.

Equivalent OCI

OCIDefineByPos

Example

The following example shows the select-list item EMPNO being defined as having position 1 and a string type.

```
DO_OCISstmtPrepare( HNDL(5), "SELECT EMPNO FROM EMP", OCI_NTV_SYNTAX );
:
```

```

:
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _STRING, 33, IS_ATTRIBUTE);

```

DO_OCIDescriptorAlloc

Allocates and initializes an Oracle 8 OCI descriptor or LOB locator.

Syntax

```
DO_OCIDescriptorAlloc( parentHandleIndex, descriptorIndex, descriptorType );
```

Parameters

Parameter	Description
parentHandleIndex	An index to an allocated Oracle 8 environment handle used as the parent handle in this call.
descriptorIndex	An index to an Oracle descriptor to be allocated and initialized.
descriptorType	The descriptor type. The Oracle 8 descriptor types used by QALoad commands are: OCI_DTYPE_LOB, OCI_DTYPE_BFILE, OCI_DTYPE_ROWID.

Equivalent OCI

OCIDescriptorAlloc

Example

```
DO_OCIDescriptorAlloc(HNDL(0), DESC(0), OCI_DTYPE_LOB);
```

DO_OCIDescriptorFree

De-allocates an Oracle 8 OCI descriptor or LOB locator.

Syntax

```
DO_OCIDescriptorFree( descriptorIndex, descriptorType );
```

Parameters

Parameter	Description
descriptorIndex	An index to an Oracle descriptor to be de-allocated.
descriptorType	The descriptor type.

Equivalent OCI

OCIDescriptorFree

Example

```
DO_OCIDescriptorFree(DESC(0), OCI_DTYPE_LOB);
```

DO_OCIEnvFreeAll

De-allocates all environment handles before the end of an OCI8 script.

Syntax

```
DO_OCIEnvFreeAll();
```

Parameters

None

Equivalent OCI

None

Example

```
DO_OCIEnvFreeAll();
:
DO_free_data();
REPORT(SUCCESS);
EXIT();
return(0);
```

DO_OCIEnvInit

Allocates and initializes an Oracle OCI8 environment handle.

Syntax

```
DO_OCIEnvInit( envHandleIndex, mode );
```

Parameters

Parameter	Description
envHandleIndex	An index to the environment handle. The mode value should be set to OCI_DEFAULT.
mode	Mode for OCI8 environment initialization.

Equivalent OCI

OCIEnvInit

Example

```
DO_OCIEnvInit(HNDL(0), OCI_DEFAULT);
```

DO_OCIEExecute

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCISmtPrepare. Note that SQL syntax errors are reported at execution time.

DO_OCIEExecute is a deprecated command. It is recommended that you use DO_OCISmtExecute instead.

Syntax

```
DO_OCIEExecute(statementHandleIndex, Iterations, mode);
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously used in the call to DO_OCISmtPrepare.

Iterations	The number of times this statement is executed for non-SELECT statements. This value can be set to 1 for SELECT statements if, and only if, all output variables were previously defined with DO_OCIDefine. This value should be set to 1.
mode	The mode for execution. The mode value should be set to the reserved word OCI_DEFAULT.

Equivalent OCI

OCIStmtExecute

Example

```
DO_OCIEExecute(HNDL(5), 1, OCI_DEFAULT);
```

DO_OCISetHandleAlloc

Allocates and initializes an Oracle 8 OCI handle.

Syntax

```
DO_OCISetHandleAlloc( parentHandleIndex, handleIndex, handleType );
```

Parameters

Parameter	Description
parentHandleIndex	An index to an allocated Oracle 8 environment handle used as the parent handle in this call.
handleIndex	An index to an Oracle handle to be allocated and initialized.
handleType	The handle type. The following handle types in Oracle 8 are used by QALoad commands: OCI_HTYPE_ERROR, OCI_HTYPE_SVCCTX, OCI_HTYPE_STMT, OCI_HTYPE_DESCRIBE, OCI_HTYPE_SERVER, OCI_HTYPE_SESSION, OCI_HTYPE_TRANS

Equivalent OCI

OCIHandleAlloc

Example

```
DO_OCISetHandleAlloc(HNDL(0), HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCISetHandleFree

De-allocates an Oracle 8 OCI handle.

Syntax

```
DO_OCISetHandleFree(handleIndex, handleType );
```

Parameters

Parameter	Description
parentHandleIndex	An index to an allocated Oracle 8 environment handle used as the parent handle in this call.

handleIndex	An index to an Oracle handle to be de-allocated.
handleType	The handle type. Following are the handle types in Oracle 8 used by QALoad commands: OCI_HTYPE_ERROR OCI_HTYPE_SVCCTX OCI_HTYPE_STMT OCI_HTYPE_DESCRIBE OCI_HTYPE_SERVER OCI_HTYPE_SESSION OCI_HTYPE_TRANS

Equivalent OCI

OCIHandleAlloc

Example

```
DO_OCIHandleFree(HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCIIinitialize

Initializes the Oracle OCI8 process environment.

This command must be issued once in a QALoad script prior to any other Oracle8 script commands, and should be outside any QALoad transactions.

Syntax

```
DO_OCIIinitialize( mode );
```

Parameters

Parameter	Description
mode	OCI8 process environment mode. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCIInitialize

Example

```
DO_OCIIinitialize(OCI_DEFAULT);
```

DO_OCILdaToSvcCtx

Toggles an Oracle 7 logon data area to an Oracle 8 service context handle.

This should be done after using DO_OCISvcCtxToLda to create Oracle 7 in a database session in Oracle 8.

Syntax

```
DO_OCILdaToSvcCtx(svcContextHandleIndex, errorHandleIndex, LdaIndex);
```

Parameters

Parameter	Description
-----------	-------------

svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
LdaIndex	An index to a Logon Data area.

Equivalent OCI

OCILdaToSvcCtx

Example

The following example shows using the Oracle7 logon data area (LDA) to create an Oracle8 service context handle using the DO_OCILdaToSvcCtx call.

```
DO_OCILdaToSvcCtx ( HNDL(4), HNDL(2), LDA(0) );
```

DO_OCILobRead

Reads a LOB into a buffer.

Syntax

```
DO_OCILobRead( svcContextHandleIndex, errorHandleIndex, lobDescriptorIndex, ReadCount,
LOBOffset, BufferLength, CharSetID, CharSetFrm );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
lobDescriptorIndex	An index to an Oracle 8 LOB locator previously allocated with a DO_OCIDescriptorAlloc call.
ReadCount	On input, the number of characters (for CLOB) or bytes (for BLOB) to be read. This variable contains the actual number of bytes or characters read after the call.
LOBOffset	On input, the absolute offset from the beginning of the LOB file. For CLOBs, this is the number of characters from the beginning. For BLOBs, it is the numbers of bytes. The first position is 1.
BufferLength	The length of the buffer. This value is specified in bytes.
CharSetID	The character set ID of the buffer data.
CharSetFrm	The character set form of the buffer data.

Equivalent OCI

OCILobRead

Example

The following example will perform a LOB read of 1024 bytes from the database.

```
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCISmtPrepare( HNDL(5), "SELECT essay FROM CLBTAB WHERE name = 'Test' " "for update",
OCI_NTV_SYNTAX );
```

QALoad 5.02

```
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _CLOB, 0, DESC(0));
DO_OCISetExecute( HNDL(5), 1, OCI_DEFAULT );
DO_OCILobRead(HNDL(3), HNDL(1), DESC(0), 1000, 1, 1024, 0, 1 );
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
```

DO_OCILobWrite

Writes the contents of a buffer into an Oracle 8 LOB.

Syntax

```
DO_OCILobWrite(svcContextHandleIndex, errorHandleIndex, lobDescriptorIndex, ReadCount,
LOBOffset, BufferLength, LOBPiece, CharSetID, CharSetFrm );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
lobDescriptorIndex	An index to an Oracle 8 LOB locator previously allocated with a DO_OCIDescriptorAlloc call.
ReadCount	On input, the number of characters (for CLOB) or bytes (for BLOB) to be written. This variable contains the actual number of bytes or characters written after the call.
LOBOffset	On input, the absolute offset from the beginning of the LOB file. For CLOBs, this is the number of characters from the beginning. For BLOBs, it is the numbers of bytes. The first position is 1.
BufferLength	The length of the buffer. This value is specified in bytes.
LOBPiece	The piece of the LOB buffer being written.
CharSetID	The LOB character set ID of the buffer data.
CharSetFrm	The LOB Character set form of the buffer data.

Equivalent OCI

OCILobWrite

Example

The following example will perform a LOB write of 1024 bytes to the database.

```
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCISetExecute( HNDL(5), "INSERT INTO CLBTAB VALUES ( 'Jack', " "EMPTY_CLOB()",
OCI_NT_V_SYNTAX );
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _CLOB, 0, DESC(0));
DO_OCISetExecute( HNDL(5), 1, OCI_DEFAULT );
DO_OCILobWrite(HNDL(3),HNDL(1),DESC(0), 1024, 1, 1024, 0, 0, 1 );
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
```


DO_OCILogoff

Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

DO_OCILogoff is a deprecated command. It is recommended that you use DO_OCILogoffEx instead.

Note: DO_OCILogoff logs off the *most recent* Oracle logon in the QALoad script. When using DO_OCILogon/DO_OCILogoff, make sure that there are no overlapping sessions.

Syntax

```
DO_OCILogoff( errorHandleIndex );
```

Note: svcContextHandleIndex is not a parameter to DO_OCILogoff. If more than one Oracle Logon is in the script, subsequent logons should be logged off with DO_OCILogoffEx.

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.

Equivalent OCI

OCILogoff

Example

```
DO_OCILogoff( HNDL(1) );
```

DO_OCILogoffEx

Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

Note: DO_OCILogoffEx will log off the Oracle logon in the QALoad script. When the script is using DO_OCILogon/DO_OCILogoffEx, QALoad playback uses OCIServerAttach, OCISessionBegin, OCIServerDetach, and OCISessionEnd calls to prevent threading issues. OCILogon and OCILogoff calls are not thread-safe.

Syntax

```
DO_OCILogoffEx( svcContextHandleIndex, errorHandleIndex );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.

Equivalent OCI

OCILogoff

Example


```
DO_OCILogoffEx ( HNDL(3), HNDL(1) );
```

DO_OCILogon

Creates a simple Oracle OCI8 logon connection and session for QALoad . Any application must log on to Oracle before performing any other Oracle operations.

For DO_OCILogon, three components must be provided:

- ! User's login ID
- ! User's password
- ! Database name as recognized by Oracle Net8 software.

 **Note:** When the script is using DO_OCILogon/DO_OCILogoffEx, QALoad uses OCIServerAttach, OCISessionBegin, OCIServerDetach, and OCISessionEnd calls to prevent threading issues. OCILogon and OCILogoff calls are not thread-safe.

Syntax

```
DO_OCILogon( envHandleIndex, errHandleIndex, svcContextHandleIndex, username, unname_len, password, passwd, dbname, dbname_len );
```

Parameters

Parameter	Description
envHandleIndex	An index to the environment handle.
errHandleIndex	An index to an allocated Oracle 8 error handle.
svcContextHandleIndex	An index to an Oracle service context handle index. This handle is automatically allocated by this call, and is automatically deallocated by a DO_OCILogoffEx call.
username	Oracle 8 user login ID.
unname_len	Character length of user login ID.
password	Oracle 8 user password for login ID.
passwd	Character length of user password.
dbname	Name of the data source to connect to.
dbname_len	Character length of data source name.

Equivalent OCI

OCILogon

Example

```
DO_OCILogon(HNDL(0), HNDL(1), "scott", 5, "tiger", 5, "oradb.world", 11);
```

DO_OCIPProcessSelectList

Fetches select-list data from an Oracle 8 database after an OCISmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

If there are no DO_OCIDefine calls before the DO_OCISmtExecute call for the select statement, the call builds up a set of internal buffers to store the returned data (otherwise done by DO_OCIDefine calls). All data is returned as ASCII strings.

Syntax

```
DO_OCIPProcessSelectList(statementHandleIndex, fetchcount );
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous calls to OCIStmtPrepare and OCIStmtExecute.
fetchCount	The count of rows to be returned per fetch loop iteration. The FetchCount value should be set to 1 unless the exact count of fetched rows is known. Note that the loop iterates until there are no more rows satisfying the SQL select request, not when the fetchCount value is reached.

Equivalent OCI

OCIStmtFetch

Example

The following example shows the DO_OCIProcessSelectList () fetch loop retrieving data after a SQL statement is executed. Note that only 1 row is fetched per loop iteration. In addition, the fetched value is processed by DO_OCIS8GetSelectData().

```
DO_OCIS8Execute( HNDL(0), HNDL(5), HNDL(1), 1, OCI_DEFAULT );
while ( DO_OCIProcessSelectList(HNDL(5), 1 ) )
{
DO_OCIS8GetSelectData( FETCH(1),COL(1), ROW(1), &FD_stmt_4_col_1_row_1, _NONE_, "", 0 );
}
```

DO_OCIProcessSelectList_EX

Fetches select-list data from an Oracle 8 database after an OCIStmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

If there are no DO_OCIDefine calls before the DO_OCIS8Execute call for the select statement, the will build up a set of internal buffers to store the returned data (otherwise done by DO_OCIDefine calls). All data is returned as ASCII strings.

DO_OCIProcessSelectList_EX extends the DO_OCIProcessSelectList macro by accommodating nested OCI8 logins. Beginning with QALoad 5.0, Compuware recommends that you use DO_OCIProcessSelectList_EX.

Syntax

```
DO_OCIProcessSelectList(statementHandleIndex, errorHandleIndex, fetchcount );
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous calls to OCIStmtPrepare and OCIStmtExecute.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
fetchCount	The count of rows to be returned per fetch loop iteration. The FetchCount value should be set to 1 unless the exact count of fetched rows is known. Note that the loop iterates until there are no more rows satisfying the SQL

select request, not when the fetchCount value is reached.

Equivalent OCI

OCIStmtFetch

Example


The following example shows the fetch loop retrieving data after a SQL statement is executed.

```
while (DO_OCIProcessSelectList_EX( HNDL(5), HNDL(2), 1 ));
{
} /*end of DO_process_select_list */
```

DO_OCIRollback

Rolls back the current Oracle8 transaction.

DO_OCIRollback is a deprecated command. It is recommended that you use DO_OCITransRollback instead.

 **Note:** DO_OCIRollback should only be used in a single-user environment. For multi-user environments, use DO_OCITransRollback.

Syntax

```
DO_OCIRollback (errorHandleIndex, CommitType);
```

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to roll back. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for QALoad scripts.

Equivalent OCI

OCITransRollback

Example

The following example shows a SQL statement being rolled back after the execute.

```
DO_OCISstmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" " , :5 )",
OCI_NTV_SYNTAX );
:
:
:
DO_OCISstmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCIRollback( HNDL(1), OCI_DEFAULT );
```

DO_OCIServerAttach

Creates a standard Oracle OCI8 database connection for QALoad . Note that individual Oracle 8 user logons are done with the DO_OCIServerAttach command.

Any application must log on to Oracle before performing any other Oracle operations. For DO_OCIServerAttach, the connect string for a database (dblink parameter) must be provided.

Syntax

```
DO_OCIServerAttach( serverHandleIndex, errorHandleIndex, dblink, dblink_len, mode );
```

Parameters

Parameter	Description
serverHandleIndex	An index to an allocated Oracle 8 service handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
dblink	Name of the data source to connect to.
dblink_len	Character length of database name.
mode	The mode of operation.

Equivalent OCI

OCIServerAttach

Example

This example shows the commands necessary to attach to an Oracle8 database.

```
DO_OCIServerAttach( HNDL(0), HNDL(2), OCI_HTYPE_SERVER);
DO_OCIServerAttach( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);
DO_OCIServerAttach(HNDL(2), HNDL(1), "oradb.world", 11, OCI_DEFAULT);
```

DO_OCIServerDetach

Detaches QALoad from the Oracle OCI8 data source connection previously attached to with the DO_OCIServerAttach command.

Note that all users must be logged off with the DO_OCISessionEnd command before this call.

Syntax

```
DO_OCIServerDetach( svcContextHandleIndex, errorHandleIndex, mode );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle previously used in a call to DO_OCIServerAttach.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
mode	Mode of operation for Oracle 8 session. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCIServerDetach

Example

This command shows the process of detaching from an Oracle 8 server and freeing the respective handles.

QALoad 5.02

```
DO_OCIServerDetach(HNDL(2), HNDL(1), OCI_DEFAULT);  
DO_OCIHandleFree( HNDL(2), OCI_HTYPE_SERVER);  
DO_OCIHandleFree( HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCISessionBegin

Creates an Oracle OCI8 logon session for QALoad to a server previously attached to with DO_OCIServerAttach.

Any application must log on to Oracle before performing any other Oracle operations.

Syntax

```
DO_OCISessionBegin( svcContextHandleIndex, errorHandleIndex, sessionHandleIndex, credt, mode  
);
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle used previously in DO_OCIServerAttach.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
sessionHandleIndex	An index to an allocated Oracle 8 session handle.
credt	Credentials for attachment to Oracle server. The Credentials value should be set to OCI_CRED_RDBMS if the username and password are required to log into the Oracle 8 database. If the database uses integrated security, set Credentials to OCI_CRED_EXT.
mode	Mode of operation for Oracle 8 session. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCISessionBegin

Example

This example shows the commands needed to begin a user session on an Oracle 8 database that has been previously attached by DO_OCIServerAttach.

```
DO_OCIHandleAlloc( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);  
DO_OCISetAttr( HNDL(3), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(1), HNDL(2));  
DO_OCIHandleAlloc( HNDL(0), HNDL(4), OCI_HTYPE_SESSION);  
DO_OCISetAttr( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),  
IS_ATTRIBUTE);  
DO_OCISetAttr( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),  
IS_ATTRIBUTE);  
DO_OCISessionBegin(HNDL(3), HNDL(1), HNDL(4), OCI_CRED_RDBMS, OCI_DEFAULT);
```

DO_OCISessionEnd

Terminates an Oracle user session previously created with the DO_OCISessionBegin command.

Syntax

```
DO_OCISessionEnd( svcContextHandleIndex, errorHandleIndex, sessionHandleIndex, mode );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle previously used in the call to DO_OCISessionBegin.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
sessionHandleIndex	An index to an allocated Oracle 8 session handle previously used in the call to DO_OCISessionBegin.
mode	The mode of operation. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCISessionEnd


Example

In the following example, the session logged on with the user name Scott and password tiger is terminated by DO_OCISessionEnd.

```
DO_OCISessionAlloc( HNDL(0), HNDL(4), OCI_HTYPE_SESSION );
DO_OCISessionAttrSet( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),
IS_ATTRIBUTE );
DO_OCISessionAttrSet( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),
IS_ATTRIBUTE );
:
:
DO_OCISessionBegin( HNDL(3), HNDL(1), HNDL(4),
OCI_CRED_RDBMS, OCI_DEFAULT );
DO_OCISessionEnd( HNDL(3), HNDL(1), HNDL(4), OCI_DEFAULT );
DO_OCISessionHandleFree( HNDL(3), OCI_HTYPE_SVCCTX );
DO_OCISessionHandleFree( HNDL(4), OCI_HTYPE_SESSION );
```

DO_OCISmtExecute

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCISmtPrepare. Note that SQL syntax errors are reported at execution time.

 **Note:** In multi-user environments, use this statement in place of DO_OCISessionExecute.

Syntax

```
DO_OCISmtExecute ( svcContextHandleIndex, statementHandleIndex, errorHandleIndex,
Iterations, mode );
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously used in the

	call to DO_OCISmtPrepare.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
Iterations	The number of times this statement is executed for non-SELECT statements. This value can be set to 1 for SELECT statements if, and only if, all output variables were previously defined with DO_OCIDefine. This value should be set to 1.
mode	The mode for execution. The mode value should be set to the reserved word OCI_DEFAULT.

Equivalent OCI

OCISmtExecute

Example

```
DO_OCISmtExecute( HNDL (3), HNDL (5),HNDL (1), OCI_DEFAULT );
```

DO_OCISmtPrepare

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

Oracle 8 SQL statements are not sent to the Oracle 8 server until execution time (handled by QALoad command DO_OCISmtExecute). SQL syntax errors are reported at execution time.

Syntax

```
DO_OCISmtPrepare(statementHandleIndex, SQLStatement, OracleSyntax);
```

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle.
SQLStatement	A pointer to a null-terminated string containing the SQL statement.
OracleSyntax	A variable flag for the parsing syntax. This value should be OCI_NTV_SYNTAX. The value for OracleLanguage should be set to OCI_NTV_SYNTAX (which defers the parsing syntax to the Oracle database) unless you want to specify a parsing syntax explicitly. Other possible values are OCI_V7_SYNTAX and OCI_V8_SYNTAX for specifying a parsing syntax based on Oracle 7 and Oracle 8, respectively.

Equivalent OCI

OCISmtPrepare

Example

The following example shows the preparation of a typical SQL statement.

```
DO_OCISmtPrepare(HNDL(5), "SELECT * FROM EMP;", OCI_NTV_SYNTAX);
```

DO_OCISmtPrepare_EX

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

Oracle 8 SQL statements are not sent to the Oracle 8 server until execution time (handled by QALoad command `DO_OCISmtExecute`). SQL syntax errors are reported at execution time.

`DO_OCISmtPrepare_EX` extends `DO_OCISmtPrepare` macro by accommodating nested OCI8 logins. Starting with QALoad 5.0, Compuware recommends that you use `DO_OCISmtPrepare_EX`.

Syntax

```
DO_OCISmtPrepareEX(statementHandleIndex, SQLStatement, errorHandleIndex, OracleSyntax);
```

Parameters

Parameter	Description
<code>statementHandleIndex</code>	An index to an allocated Oracle 8 statement handle.
<code>SQLStatement</code>	A pointer to a null-terminated string containing the SQL statement.
<code>errorHandleIndex</code>	An index to an allocated Oracle 8 error handle.
<code>OracleSyntax</code>	A variable flag for the parsing syntax. This value should be <code>OCI_NTV_SYNTAX</code> . The value for <code>OracleLanguage</code> should be set to <code>OCI_NTV_SYNTAX</code> (which defers the parsing syntax to the Oracle database) unless you want to specify a parsing syntax explicitly. Other possible values are <code>OCI_V7_SYNTAX</code> and <code>OCI_V8_SYNTAX</code> for specifying a parsing syntax based on Oracle 7 and Oracle 8, respectively.

Equivalent OCI

`OCISmtPrepare`

Example

The following example shows the preparation of a typical SQL statement.

```
DO_OCISmtPrepare_EX(HNDL(5), "SELECT * FROM EMP;", HNDL(2), OCI_NTV_SYNTAX );
```

DO_OCISvcCtxToLda

Toggles an Oracle 8 service context handle to an Oracle 7 logon data area. This allows Oracle 7 cursors to be created in a database session created in Oracle 8.

Syntax

```
DO_OCISvcCtxToLda(svcContextHandleIndex, errorHandleIndex, LdaIndex);
```

Parameters

Parameter	Description
<code>svcContextHandleIndex</code>	An index to the current allocated Oracle 8 service context handle.
<code>errorHandleIndex</code>	An index to an allocated Oracle 8 error handle.
<code>LdaIndex</code>	An index to a Logon Data area.

Equivalent OCI

`OCISvcCtxToLda`

Example

The following example shows toggling the Oracle8 service context handle to an Oracle7 logon data area (LDA) using the DO_OCISvcCtxToLda call.

```
DO_OCISvcCtxToLda(HNDL(4), HNDL(2), LDA(0));
```

DO_OCITransCommit

Commits the current Oracle 8 transaction. A commit should be performed after all relevant SQL statements have been processed.

Syntax

```
DO_OCITransCommit (svcContextHandleIndex, errorHandleIndex, CommitType);
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to commit. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for QALoad scripts.

Example

The following example shows an insert transaction being committed after the execute.

```
DO_OCISstmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" ", :5 )",
OCI_NTV_SYNTAX );
:
:
:
DO_OCISstmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCITransCommit ( HNDL(3), HNDL(1), OCI_DEFAULT );
```

DO_OCITransRollback

Rolls back the current Oracle 8 transaction.

Syntax

```
DO_OCITransRollback (svcContextHandleIndex, errorHandleIndex, CommitType);
```

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to roll back. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for QALoad scripts.

Equivalent OCI

OCITransRollback

Example

```
DO_OCISstmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" " , :5 )",
OCI_NTV_SYNTAX );
:
:
:
DO_OCISstmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCITransRollback ( HNDL(3), HNDL(1), OCI_DEFAULT );
```

Oracle Forms Server

Oracle Forms Server Index

Add

Adds a property to the current message.

AddListboxValue

Used with list box objects to send a specific list box value to the Oracle Forms Server.

AlertDialog

This method constructs a new OracleFormsMsg for this type of control.

AppTimer

This method constructs a new OracleFormsMsg for this type of control.

BlockScroller

This method constructs a new OracleFormsMsg for this type of control.

Button

This method constructs a new OracleFormsMsg for this type of control.

CancelQueryDialog

This method constructs a new OracleFormsMsg for this type of control.

CfmOLE

This method constructs a new OracleFormsMsg for this type of control.

CfmVBX

This method constructs a new OracleFormsMsg for this type of control.

Checkbox

This method constructs a new OracleFormsMsg for this type of control.

CheckForErrorMsgs

Indicates whether messages from the Oracle Forms Server are checked for error messages.

Connect

This method establishes a connection with the Oracle Forms Server.

Disconnect

Closes the connection to the Oracle Forms Server.

DisplayErrorDialog

This method constructs a new OracleFormsMsg for this type of control.

[DisplayList](#)

This method constructs a new OracleFormsMsg for this type of control.

[EditorDialog](#)

This method constructs a new OracleFormsMsg for this type of control.

[FormCanvas](#)

This method constructs a new OracleFormsMsg for this type of control.

[FormStatusBar](#)

This method constructs a new OracleFormsMsg for this type of control.

[FormWindow](#)

This method constructs a new OracleFormsMsg for this type of control.

[GetControlValue](#)

This method returns the current contents of the control. Support is limited to text fields.

[HelpDialog](#)

This method constructs a new OracleFormsMsg for this type of control.

[HTTPConnectToFormsServlet](#)

Allows the client to connect to the 9i Forms Servlet. Automatic checkpointing is performed here.

[HTTPConnectToListenerServlet](#)

Allows the client to connect to the 9i Listener Servlet. Automatic checkpointing is performed here.

[HTTPInitialFormsConnect](#)

Allows the client to establish connection with the Forms Server via the Listener Server. Automatic checkpointing is performed here.

[HTTPReceiveMessage](#)

Allows the client to send the HTTP request (containing Forms messages) and receive the HTTP reply. Automatic checkpointing is done here. Forms messages from the HTTP reply are individually processed.

[HTTPDoSSLHandshake](#)

Enables QALoad to do an SSL handshake prior to an SSL-enabled Forms connection.

[HTTPSetHdrProperty](#)

Defines the HTTP connection properties that the client will use to connect to the Forms Servlet and the Listener Servlet. The properties given in the script are User-Agent, Host, Accept, Connection, and Content-type.

[HTTPSetListenerServletParms](#)

Defines the HTTP connection parameters that the client will use to connect to the 9i Listener Servlet. The parameters given in the script are: ifcmd, ifhost, and ifip.

[HTTPSetURL](#)

Allows the client to define the Listener Servlet for applications running Forms 6i patch 4+, prior to the execution of HTTPInitialFormsConnect.

[HTTPXmitMsg](#)

Allows the client to accumulate the Forms messages before sending the HTTP request.

[HTTPXmitTerminalMessage](#)

Allows QALoad to prepare the HTTP request that contains the current set of Forms messages. Preparation includes establishing a new HTTP connection, stream, and content size, as well as encrypting and writing to the HTTP stream.

[IconicButton](#)

This method constructs a new OracleFormsMsg for this type of control.

[ImageItem](#)

This method constructs a new OracleFormsMsg for this type of control.

[JavaContainer](#)

This method constructs a new OracleFormsMsg for this type of control.

[ListValuesDialog](#)

This method constructs a new OracleFormsMsg for this type of control.

[LogComment](#)

If logging is enabled, this method will record its string parameter as a comment in the log file. These comments can be very useful when utilizing the log file for debugging.

[Logging](#)

This method enables logging of client to server traffic during script playback.

[LogonDialog](#)

This method constructs a new OracleFormsMsg for this type of control.

[MenuInfo](#)

This method constructs a new OracleFormsMsg for this type of control.

[MenuParametersDialog](#)

This method constructs a new OracleFormsMsg for this type of control.

[ofsActivateListItem](#)

Adds the TList_Activated property to the current message. Tlist_Activated property indicates user selection of an item in a List control.

[ofsActivateTreeItem](#)

Adds the Event_Activated property of a Tree control to the current message. Event_Activated property indicates user selection of an item in a Tree control.

[ofsActivateWindow](#)

Adds the Window_Activate property (with Enabled attribute) to the current message.

[ofsClickButton](#)

Adds the Pressed property of a Button control to the current message.

[ofsClickTextFieldItem](#)

Adds the Pressed property associated with a Text Field control to the current message.

[ofsClosePopList](#)

Adds the List_Closed property of a PopList control to the current message.

[ofsCloseWindow](#)

Adds the Window_Close property (with Enabled attribute) to the current message.

[ofsCollapseTreeltem](#)

Adds the Event-Collapsed property of a Tree control to the current message.

[ofsColorAdd](#)

Adds the Color_Add property to the current message.

[ofsConnectToSocket](#)

Establishes a socket-mode connection to the Oracle Forms Server.

[ofsDeActivateWindow](#)

Adds the Window_Activate property (with Disabled attribute) to the current message.

[ofsDefineTreeNode](#)

Adds the Node_ID property of a Tree control to the current message. Node_ID property defines the relative position of the tree item, counting nested tree items.

[ofsDefineTreeNodeOffset](#)

Adds the Node_Offset property of a Tree control to the current message. Node_Offset defines the relative position of the tree item, excluding nested tree items.

[ofsDelconifyWindow](#)

Adds the Window_Iconified property (with Disabled attribute) to the current message.

[ofsDeSelectItem](#)

Adds the Value property (with Disabled attribute) to the current message.

[ofsDeSelectTreeEvent](#)

Adds the Event_DeSelect property of a Tree control to the current message. This statement indicates the application is moving from an internal processing event that is associated with a tree item.

[ofsEdit](#)

Adds the Value property to the current message. The property is associated with a Text Field control.

[ofsExpandTreeltem](#)

Adds the Event_Expanded property of a Tree control to the current message. The Event_Expanded property indicates a Tree control item being expanded.

[ofsFindLOVValue](#)

Adds the LOV_Find_Value property of a List of Values control to the current message. The statement indicates the user is searching for an item in a List of Values control.

[ofsFocus](#)

Adds the Focus property (with Enabled attribute) to the current message.

[ofsGetServerData](#)

Returns the Forms data from the server reply.

[ofsHideWindow](#)

Adds the Visible property (with Disabled attribute) to the current message.

[ofsHTTPDisconnect](#)

Closes the current HTTP connection to the Forms Listener servlet.

[ofsHTTPDoSSLHandshake](#)

Establishes an SSL socket connection and starts an SSL handshake.

[ofsHTTPSetHdrProperty](#)

Establishes the HTTP headers to use for connecting to the Forms servlet and listener servlet.

[ofsHTTPSetListenerServletParms](#)

Sets the Forms Listener Servlet parameters prior to connection.

[ofsHTTPConnectToFormsServlet](#)

Opens an HTTP connection to the Forms servlet responsible for initiating a Forms applet instance.

[ofsHTTPConnectToListenerServlet](#)

Opens an HTTP connection to the Forms Listener servlet responsible for starting an instance of the Forms run time process.

[ofsHTTPInitialFormsConnect](#)

Opens an HTTP connection to the Forms Listener servlet and posts the initial Forms handshake information.

[ofsIconifyWindow](#)

Adds the Window_Iconified property (with Enabled attribute) to the current message.

[ofsIndexKey](#)

Adds the Index_Key property to the current message.

[ofsIndexSKey](#)

Adds the Index_SKey property to the current message.

[ofsInitSessionCmdLine](#)

Adds the INITIAL_CMDLINE property to the current message.

[ofsInitSessionTimeZone](#)

Adds the Time_Zone property to the current message.

[ofsListItemValue](#)

Adds the List_Item property of a PopList or a TList control to the current message.

[ofsLoadValue](#)

Loads the values of a byte array or a string array associated with a GUI control.

[ofsLOVRequestRow](#)

Adds the LOV_REQUEST_ROW property to the current message.

[ofsLOVSelection](#)

Adds the LOV_SELECTION property to the current message.

[ofsMenuParamDlgOK](#)

Adds the MENUPARAM_DLGOK property to the current message. This statement defines the text in the menu param dialog control.

[ofsOpenWindow](#)

Adds the Window_Open property (with Disabled attribute) to the current message.

[ofsRemoveFocus](#)

Adds the Focus property (with Disabled attribute) to the current message.

[ofsSetCursorPosition](#)

Adds the Cursor_Position property of a Text Field control to the current message.

[ofsSetErrorDialogTitle](#)

Adds the DISPLAYERRORDIALOG_TITLE property to the current message.

[ofsSetFontName](#)

Adds the Font_Name property to the current message.

[ofsScroll](#)

Adds the Block_Scroller property to the current message.

[ofsScrollSize](#)

Adds the Block_Scroller_Size property to the current message.

[ofsSelectItem](#)

Adds the Value property (with Enabled attribute) to the current Message.

[ofsSelectMenuItem](#)

Adds the Menu_Event property to the current message.

[ofsSelectTreeEvent](#)

Adds the Selected_Event property of a Tree Control to the current message.

[ofsSendRecv](#)

Sends the client request as Forms messages to the Forms server, gets the server response, and reads the responses as Forms messages.

[ofsServerSideDisconnect](#)

Disconnects QALoad's socket connection to the server-side code. The server-side code intercepts the messages between QALoad and the Forms Listener servlet.

[ofsSetColorDepth](#)

Adds the Color_Depth property to the current message.

[ofsSetDisplaySize](#)

Adds the Display_Size property to the current message.

[ofsSetExpectedServerMsg](#)

Enables the script to continue if a known error or warning message is received from the server.

[ofsSetFontName](#)

Adds the Font_Name property to the current message.

[ofsSetFontSize](#)

Adds the Font_Size property to the current message.

[ofsSetFontStyle](#)

Adds the Font_Style property to the current message.

[ofsSetFontWeight](#)

Adds the Font_Weight property to the current message.

[ofsSetICXTicket](#)

Sets the value of the ICX ticket for the current Oracle Applications login. The statement is used only in a Universal OFS-WWW session, as a replacement for the OracleAppsLogin() statement.

[ofsSetInitialVersion](#)

Adds the Initial_Version property to the current message.

[ofsSetJavaContainerArgName](#)

Adds the JAVACONTAINER_ARG_NAME property to the current message.

[ofsSetJavaContainerArgValue](#)

Adds the JAVACONTAINER_ARG_VALUE property to the current message.

[ofsSetJavaContainerEvent](#)

Adds the JAVACONTAINER_ARG_EVENT property to the current message.

[ofsSetLogonDatabase](#)

Adds the LOGON_DATABASE property to the current message.

[ofsSetLogonPassWord](#)

Adds the LOGON_PASSWORD property to the current message.

[ofsSetLogonUserName](#)

Adds the LOGON_USERNAME property to the current message.

[ofsSetNoRequiredVAList](#)

Adds the Required_VA_List property (with Disabled attribute) to the current message.

[ofsSetPropertyBoolean](#)

Adds the generic boolean property (with Enabled attribute) to the current message.

[ofsSetPropertyByte](#)

Adds the generic byte property to the current message.

[ofsSetPropertyByteArray](#)

Adds the generic byte array property to the current message.

[ofsSetPropertyCharacter](#)

Adds the generic Character property to the current message.

[ofsSetPropertyDate](#)

Adds the generic Date property to the current message.

[ofsSetPropertyFloat](#)

Adds the generic Float property to the current message.

[ofsSetPropertyInteger](#)

Adds the generic Integer property to the current message.

[ofsSetPropertyPoint](#)

Adds the generic Point property to the current message.

[ofsSetPropertyRectangle](#)

Adds the generic Rectangle property to the current message.

[ofsSetPropertyString](#)

Adds the generic String property to the current message.

[ofsSetPropertyStringArray](#)

Adds the generic String array property to the current message.

[ofsSetPropertyVoid](#)

Adds the generic Void property to the current message.

[ofsSetRequiredVAList](#)

Adds the Required_VA_List property (with Enabled attribute) to the current message.

[ofsSetRunOptions](#)

Sets the runtime values for CONNECT TYPE, HEARTBEAT, LOGGING (to replay capture file) and CHECK SERVER MESSAGES.

[ofsSetScaleInfo](#)

Adds the Scale property to the current message.

[ofsSetScreenResolution](#)

Adds the Screen Resolution property to the current message.

[ofsSetSelection](#)

Adds the Selection property of a Text Field control to the current message.

[ofsSetServletMode](#)

Creates a socket connection to the server-side code which communicates with the Forms Listener Servlet.

[ofsSetServerFailedMsg](#)

Enables QALoad to fail playback based on the user-entered string and filter parameters.

[ofsSetValue](#)

Adds a generic Value property to the current message.

[ofsSetWindowLocation](#)

Adds the Location property of a Window control to the current message.

[ofsSetWindowSize](#)

Adds the Size property of a Window control to the current message.

[ofsShowWindow](#)

Adds the Visible property (with Enabled attribute) to the current message.

[ofsSocketDisconnect](#)

Closes the connection of a socket-mode playback.

[ofsStartSubMessage](#)

Adds a sub-message to the current message.

[ofsTabControlTopPage](#)

Adds the TabControl_Top_Page property to the current message.

[ofsUnsetPropertyBoolean](#)

Adds a generic Boolean property (with Disabled attribute) to the current message.

[OracleAppsLogin](#)

This method simulates an Oracle Applications 11i login and retrieves the icx_ticket associated with that login. It should be performed once per virtual user.

OracleForms

The constructor for the OracleForms class. This should be classed once per script.

OracleFormsMsg

This method constructs a new OracleFormsMsg.

PopList

This method constructs a new OracleFormsMsg for this type of control.

PopupHelp

This method constructs a new OracleFormsMsg for this type of control.

PromptList

This method constructs a new OracleFormsMsg for this type of control.

RadioButton

This method constructs a new OracleFormsMsg for this type of control.

Runform

This method constructs a new OracleFormsMsg for this type of control.

SetExpectedServerMsg

This method allows the script to continue processing if a known error or warning message is received from the server.

SetHeartbeat

Sets the interval time for the transmission of internal heartbeat messages to the server. Without this statement, Playback takes the default 2 minutes. If the parameter is set to 0, transmission of heartbeat messages is suppressed.

SetProxy

This method is only used if the script is communicating with the Oracle Forms Server via HTTP. It enables the HTTP traffic to be routed through a proxy server.

TabControl

This method constructs a new OracleFormsMsg for this type of control.

TextArea

This method constructs a new OracleFormsMsg for this type of control.

TextField

This method constructs a new OracleFormsMsg for this type of control.

Tlist

This method constructs a new OracleFormsMsg for this type of control.

Tree

This method constructs a new OracleFormsMsg for this type of control.

XmitMsg

Sends a message to the Oracle Forms Server.

XmitTerminalMessage

Sends a terminal message to the Oracle Forms Server.

Add

Adds a property to the current message.

Syntax

```
Add( propertyId, data )
```

Return Value

Void

Parameters

Parameter	Description
PropertyId int	The property code for this data. There are hundreds of different property codes. The QALoad Script Development Workbench creates a list of readable property codes that are used in the script.
data Object	This parameter is a Java object.

Exceptions

None.

Example

```
oracleFormsMsg1.Add( PROPERTY_FOCUS, new Boolean( true ) );
oracleFormsMsg1.Add( PROPERTY_SELECTION, new
    java.awt.Point(13,13) );
```

AddListboxValue

Used with list box objects to send a specific list box value to the Oracle Forms Server.

Each value in a list box carries with it an index assigned by the forms server. When the end-user selects an entry, that index is sent to the server.

When data is received from the Oracle Forms Server, the script code keeps a table of index/value pairs for each list box object. This method searches that table and automatically inserts the appropriate index for the value specified in the script.

Syntax

```
Add( propertyId, value )
```

Return Value

Void

Parameters

Parameter	Description
PropertyId int	The property code for this data. There are hundreds of different property codes. The QALoad Script Development Workbench creates a list of readable property codes that are used in the script.

Value String	List box value.
--------------	-----------------

Exceptions

QALoad Exception("Listbox value not found. Value: " + <value>)

Example

```
oracleFormsMsg1.AddListboxValue( PROPERTY_VALUE, "Value 4" );
```

AlertDialog

Constructs a new OracleFormsMsg for this type of control.

Syntax

```
AlertDialog( handlerID );
AlertDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.AlertDialog( CONTROL_001 );
oracleFormsMsg1 = oracleForms.AlertDialog( "CONTROLabc" );
```

AppTimer

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
AppTimer( handlerID );
AppTimer( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.AppTimer( CONTROL_001 );
oracleFormsMsg1 = oracleForms.AppTimer( "CONTROLabc" );
```

BlockScroller

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
BlockScroller( handlerID );
BlockScroller( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.BlockScroller( CONTROL_001 );
oracleFormsMsg1 = oracleForms.BlockScroller( "CONTROLabc" );
```

Button

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
Button( handlerID );
Button( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.Button( CONTROL_001 );
oracleFormsMsg1 = oracleForms.Button( "CONTROLabc" );
```

CancelQueryDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
CancelQueryDialog( handlerID );
CancelQueryDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.CancelQueryDialog ( CONTROL_001 );
oracleFormsMsg1 = oracleForms.CancelQueryDialog ( "CONTROLabc" );
```

CfmOLE

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
CfmOLE( handlerID );
CfmOLE( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.CfmOLE( CONTROL_001 );
oracleFormsMsg1 = oracleForms.CfmOLE( "CONTROLabc" );
```

CfmVBX

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
CfmVBX( handlerID );
CfmVBX( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```


Example

```
oracleFormsMsg1 = oracleForms.CfmVBX( CONTROL_001 );
oracleFormsMsg1 = oracleForms.CfmVBX( "CONTROLabc" );
```

Checkbox

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
Checkbox( handlerID );
Checkbox( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.Checkbox( CONTROL_001 );
oracleFormsMsg1 = oracleForms.Checkbox( "CONTROLabc" );
```

CheckForErrorMsgs

Indicates whether messages from the Oracle Forms Server are checked for error messages.

Each incoming message from the server is checked for warnings or errors. If error checking is enabled and such a message is received, then an exception is thrown.

Syntax

```
CheckForErrorMsgs( bFlag )
```

Return Value

Void

Parameters

Parameter	Description
BFlag Boolean	True to enable checking, false to disable it.

Exceptions

None thrown.

Example

```
oracleForms.CheckForErrorMsgs( true );
// Checking for errors
```

Connect

This method establishes a connection with the Oracle Forms Server.

Syntax

```
Connect( statement, hostname, port, type );
```

Return Value

Void

Parameters

Parameter	Description
Statement int	A number used for script debugging and error reporting.
Hostname String	Host name or IP address of the Oracle Forms Server.
Port int	Port number to connect to on the server.
Type int	Type of connection. Must be one of the following two constants: WF_SOCKET WF_HTTP

Exceptions

```
QALoad Exception( "Connect called, with an already established connection.")
QALoad Exception( "Unknown host: " )
QALoad Exception( "IOException called when making a connection.")
```

Example

```
oracleForms.Connect( 1, "192.168.0.96", 9000, OracleForms.WF_SOCKET );
```

Disconnect

Closes the connection to the Oracle Forms Server.

Syntax

```
Disconnect( statement )
```

Return Value

Void

Parameters

Parameter	Description
Statement int	A number used for script debugging and error reporting.

Exceptions

QALoad Exception("IOException called when closing server socket.")

Example

```
oracleForms.Disconnect( 48 );
```

DisplayErrorDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
DisplayErrorDialog( handlerID );
DisplayErrorDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>'");

Example

```
oracleFormsMsg1 = oracleForms.DisplayErrorDialog ( CONTROL_001 );
oracleFormsMsg1 = oracleForms.DisplayErrorDialog ( "CONTROLabc" );
```

DisplayList

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
DisplayList( handlerID );
DisplayList( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or

	type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.DisplayList( CONTROL_001 );
oracleFormsMsg1 = oracleForms.DisplayList( "CONTROLabc" );
```

EditorDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
EditorDialog( handlerID );
EditorDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.EditorDialog( CONTROL_001 );
oracleFormsMsg1 = oracleForms.EditorDialog( "CONTROLabc" );
```

FormCanvas

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
FormCanvas( handlerID );
FormCanvas( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.FormCanvas( CONTROL_001 );
oracleFormsMsg1 = oracleForms.FormCanvas( "CONTROLabc" );
```

FormStatusBar

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
FromStatusBar( handlerID );
FromStatusBar( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.FromStatusBar( CONTROL_001 );
oracleFormsMsg1 = oracleForms.FromStatusBar( "CONTROLabc" );
```

FormWindow

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
FormWindow( handlerID );
FormWindow( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.FormWindow( CONTROL_001 );
oracleFormsMsg1 = oracleForms.FormWindow( "CONTROLabc" );
```

GetControlValue

This method returns the current contents of the control. Support is limited to text fields.

Syntax

```
GetControlValue( handlerId )
GetControlValue( controlName )
```

Return Value

String

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
String orderNumber = oracleForms.GetControlValue ( "OrderField" );
String orderNumber = oracleForms.GetControlValue ( TextField_001 );
```

HelpDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
HelpDialog( handlerID );
HelpDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ( "No match for control name: <control_name>'");
```

Example

```
oracleFormsMsg1 = oracleForms.HelpDialog( CONTROL_001 );
oracleFormsMsg1 = oracleForms.HelpDialog( "CONTROLabc" );
```

HTTPConnectToFormsServlet

Allows the client to connect to the 9i Forms Servlet. Automatic checkpointing is performed here.

Syntax

```
HTTPConnectToFormsServlet( stmtNum, formsServletURL, connectType );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error.
formsServletURL string	Captured Forms Servlet name and connection parameters.
connectType int	Type of connection; always the integer WF_HTTP.

Exceptions

QALoad Exception("OFS error - HTTPConnectToFormsServlet: failed to connect to the Forms Servlet")

Example

```
oracleForms.HTTPConnectToFormsServlet( 5,
"http://ntsap45b:7779/forms90/f90ervlet?ifcmd=startsession HTTP/1.1", OracleForms.WF_HTTP
);
```

HTTPConnectToListenerServlet

Allows the client to connect to the 9i Listener Servlet. Automatic checkpointing is performed here.

Syntax

```
HTTPConnectToListnerServlet( stmtNum, ServletURL );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error.
servletURL string	Captured name of the Forms 9i Listener Servlet.

Exceptions

QALoad Exception("OFS error - HTTPConnectToListenerServlet: failed to connect to the Listener Servlet")

Example

```
oracleForms.HTTPConnectToListenerServlet( 7, "http://ntsap45b:7779/forms90/l90ervlet" );
```

HTTPInitialFormsConnect

Allows the client to establish connection with the Forms Server via the Listener Server. Automatic checkpointing is performed here.

Syntax

```
HTTPInitialFormsConnect( stmtNum, connectType );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.
connectType int	Type of connection; always the integer WF_HTTP.

Exceptions

QALoad Exception("OFS error - HTTPInitialFormsConnect: failed to connect to the Forms Server")

Example

```
oracleForms.HTTPInitialFormsConnect( 9, OracleForms.WF_HTTP );
```

HTTPReceiveMessage

Allows the client to send the HTTP request containing Forms messages and receive the HTTP reply. Automatic checkpointing is done here. Forms messages from the HTTP reply are individually processed.

Syntax

```
HTTPReceiveMessage( stmtNum );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.

Exceptions

QALoad Exception("OFS error - HTTPReceiveMessage: failed to read HTTP reply message.")

Example

```
oracleForms.HTTPReceiveMessage( 9 ); //Statement # = 9
```

HTTPSDoSSLHandshake

Enables QALoad to do an SSL handshake prior to an SSL-enabled Forms connection.

Syntax

```
HTTPSDoSSLHandshake(int stmtNum);
```

Return Value

Void

Parameters

Parameter	Description
stmtNum	Script statement number that is used for script debugging and error reporting.

Exceptions

QALoad Exception("OFS - HTTPSDoSSLHandShake")

Example

```
oracleForms.HTTPSDoSSLHandshake(5);
/* The number 5 is used to display that statement number in the ValidateDebug Trace window
*/
```

HTTPSetHdrProperty

Defines the HTTP connection properties that the client uses to connect to the Forms Servlet and the Listener Servlet.

The properties given in the script are User-Agent, Host, Accept, Connection, and Content-type.

Syntax

```
HTTPSetHdrProperty( stmtNum, propertyName, propertyValue );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.
propertyName string	HTTP connection property name.
propertyValue string	Captured value of the connection property.

Exceptions

```
QALoad Exception( "OFS error - HTTPSetHdrProperty: failed to set request header property" )
```

Example

```
oracleForms.HTTPSetHdrProperty( 1, "User-Agent", "Java1.3.1.9" );
```

HTTPSetListenerServletParms

Defines the HTTP connection parameters that the client uses to connect to the 9i Listener Servlet.

The parameters given in the script are: ifcmd, ifhost, and ifip.

Syntax

```
HTTPSetListnerServletParms( stmtNum, servletParams );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int:	Script statement number used for script debugging and error reporting.
servletParams string	Captured servlet parameters; during playback, the ifhost and ifip values must be the host name and IP address of the Playback machine.

Exceptions

None

Example

```
oracleForms.HTTPSetListenerServletParms( 6,
"?ifcmd=getinfo&ifhost=sfa10453&ifip=172.22.24.91" );
```

HTTPSetURL

Allows the client to define the Listener Servlet for applications running Forms 6i patch 4+, prior to the execution of HTTPInitialFormsConnect.

Syntax

```
HTTPSetURL( stmtNum, servletURL );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.
servletURL string	Captured name of the Forms Listener Servlet (Forms 6i patch 4+) plus the HTTP version.

Exceptions

None

Example

```
oracleForms.HTTPSetURL( 5,
"http://ntsap45b:7782/servlet/oracle.forms.servlet.ListenerServlet HTTP/1.0" );
```

HTTPXmitMsg

Allows the client to accumulate the Forms messages before sending the HTTP request.

Syntax

```
HTTPXmitMsg( stmtNum, msgObject );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.
msgObject string	Script's current message object.

Exceptions

```
QALoad Exception( "OFS error - HTTPXmitMsg:Exception called when adding msg to msg array" )
```

Example

```
oracleForms.HTTPXmitMsg( 8, oracleForms.HTTPXmitMsg1 );
// Statement # = 8
```

HTTPXmitTerminalMessage

Allows QALoad to prepare the HTTP request that contains the current set of Forms messages.

QALoad 5.02

Preparation includes establishing a new HTTP connection, stream, and content size, as well as encrypting and writing to the HTTP stream.

Syntax

```
HTTPXmitTerminalMessage( stmtNum, respCode );
```

Return Value

Void

Parameters

Parameter	Description
stmtNum int	Script statement number used for script debugging and error reporting.
respCode int	Captured terminal - message's response code indicating the type of action to take. A value of 1 indicates that the messages should be acted on. A value of 3 indicates that the client requests to disconnect from the server.

Exceptions

```
QALoad Exceptions( "OFS error - HTTPXmitTerminalMessage: failed to prep HTTP request" )
```

Example

```
oracleForms.HTTPXmitTerminalMessage( 9, 1 ); //Statement #=9 );
```

IconicButton

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
IconicButton( handlerID );  
IconicButton( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ( "No match for control name: <control_name>'");
```

Example

```
oracleFormsMsg1 = oracleForms.IconicButton( CONTROL_001 );  
oracleFormsMsg1 = oracleForms.IconicButton( "CONTROLabc" );
```

ImageItem

Constructs a new OracleFormsMsg for this type of control.

Syntax

```
ImageItem( handlerID );
ImageItem( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.ImageItem( CONTROL_001 );
oracleFormsMsg1 = oracleForms.ImageItem( "CONTROLabc" );
```

JavaContainer

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
JavaContainer( handlerID );
JavaContainer( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.JavaContainer( CONTROL_001 );
oracleFormsMsg1 = oracleForms.JavaContainer( "CONTROLabc" );
```

ListValuesDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
ListValuesDialog( handlerID );
ListValuesDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.ListValuesDialog ( CONTROL_001 );
oracleFormsMsg1 = oracleForms.ListValuesDialog ( "CONTROLabc" );
```

LogComment

If logging is enabled, this method records its string parameter as a comment in the log file.

These comments can be very useful when utilizing the log file for debugging.

Syntax

```
LogComment( strComment )
```

Return Value

Void

Parameters

Parameter	Description
StrComment String	Comment to log.

Exceptions

None

Example

```
oracleForms.LogComment( "Enter customer here." );
```

Logging

This method enables logging of client-to-server traffic during script playback.

The log file is named: OFSLOG_<scriptName>_<vu #>.cap, and is stored in the \QALoad directory.

Syntax

```
Logging( flag )
```

Return Value

Void

Parameters

Parameter	Description
flag Boolean	Valid values are: true: Enable logging false: Disable logging

Exceptions

```
QALoad Exception("IOException called when enabling logging.");
```

Example

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

LogonDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
LogonDialog( handlerID );
```

```
LogonDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded

with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.LogonDialog( CONTROL_001 );
oracleFormsMsg1 = oracleForms.LogonDialog( "CONTROLabc" );
```

MenuInfo

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
MenuInfo( handlerID );
MenuInfo( controlName );
```

Return Value

An OracleFormsMsg

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.MenuInfo( CONTROL_001 );
oracleFormsMsg1 = oracleForms.MenuInfo( "CONTROLabc" );
```

MenuParametersDialog

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
MenuParametersDialog( handlerID );
MenuParametersDialog( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
-----------	-------------

HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.MenuParametersDialog ( CONTROL_001 );
oracleFormsMsg1 = oracleForms.MenuParametersDialog ( "CONTROLabc" );
```

ofsActivateListItem

Adds the TList_Activated property to the current message. Tlist_Activated property indicates user selection of an item in a List control.

Syntax

```
void ofsActivateListItem(const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *sValue );
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
sValue	The positional value of the activated List item.

Example

```
//In the example below, the List item is defined,
//and then selected using the statement
//ofsActivateListItem. The value 7 indicates
//that the item is the 7th List item.

ofsListItemValue( "TLIST", 118, OFS_ENDMSG, 131, "7" ); /*Item value =      Material
      Transactions*/

ofsSendRecv(1 );
```

QALoad 5.02

```
:  
:  
ofsActivateListItem( "TLIST", 118, OFS_ENDMSG, 341, "7" );  
ofsSendRecv(1 );
```

ofsActivateTreeItem

Adds the Event_Activated property of a Tree control to the current message. Event_Activated property indicates user selection of an item in a Tree control. The selected item is associated with internal processing events.

Syntax

```
void ofsActivateTreeItem(const char *sHandlerName, int ControlID, int ActionType, int  
PropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The positional value of the activated Tree item.

Example

```
//The statement ofsActivateTreeItem is  
//similar to ofsActivateListItem  
//but internal processing events occur  
//when it is executed. A Tree item is a  
//List Item that is associated with an event.  
//In this example, item 4 (named "Sample Event1")  
//in Tree Control ID 118 is selected.  
  
ofsListItemValue( "TLIST", 118, OFS_ENDMSG, 131, "4" ); /*Item value = Sample Event1*/  
ofsSendRecv(1 );  
  
.  
.  
.  
  
ofsActivateTreeItem( "Test Tree", 118, OFS_ENDMSG, 491, "4" );
```

ofsActivateWindow

Adds the Window_Activate property (with Enabled attribute) to the current message. The Window_Activate (with Enabled attribute) property indicates the opening of a new window.

Syntax

```
void ofsActivateWindow( const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//This example indicates the window "Oracle
//Applications" being displayed as the top window,
//then the window is activated for use.

ofsShowWindow( "Oracle Applications", 32, OFS_ENDMSG, 173 );
ofsActivateWindow( "Oracle Applications", 32, OFS_ENDMSG, 247 );
ofsFocus( "TEXTFIELD", 75, OFS_ENDMSG, 174 );
ofsSendRecv(2 );
```

ofsClickButton

Adds the Pressed property of a Button control to the current message. This statement indicates a button click activity.

Syntax

```
void ofsClickButton(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.

PropertyID	Oracle-designated ID for the property being added.
------------	--

Example

```
//In this example, control ID 52 represents the button that is clicked.
ofsClickButton( "BUTTON", 52, OFS_ENDMSG, 325 );
ofsSendRecv(1 );
```

ofsClickTextFieldItem

Adds the Pressed property associated with a Text Field control to the current message. This statement indicates an activity in which focusing on a Text Field item enables the user to click a button that triggers internal processing events.

Syntax

```
void ofsClickTextFieldItem(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message. OFS_STARTSUBMSG: Add the property of the succeeding nested message to the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, the location of Text Field
//control 274 is defined using ofsSetSelection.

//The Browse button embedded in Text Field control
//274 is clicked. The click, simulated by
//ofsClickTextFieldItem, deactivated the currently
//opened window "Find Material Transactions"
//(control 179) and also triggered Custom Control
//1367 to act as the top window.

ofsSetSelection( "TEXTFIELD", 274, OFS_ENDMSG, 195, 0, 0);
ofsClickTextFieldItem( "TEXTFIELD", 274, OFS_ENDMSG, 325 );
ofsSendRecv(1 );
ofsSendRecv(1 );
ofsDeActivateWindow( "Find Material Transactions ", 179, OFS_ENDMSG, 247 );
ofsSendRecv(1 );

ofsSetPropertyInteger( "CUSTOMCONTROL", 1367, OFS_ADD, 2601, "91" );
ofsSetPropertyInteger( "CUSTOMCONTROL", 1367, OFS_ADD, 2600, "0" );
```

```
ofsSetPropertyInteger( "CUSTOMCONTROL", 1367, OFS_ADD, 2600, "0" );
ofsSetPropertyString( "CUSTOMCONTROL", 1367, OFS_ENDMSG, 2600, "xxx" );
ofsSendRecv(1 );
```

ofsClosePopList

Adds the List_Closed property of a PopList control to the current message. The List_Closed property indicates a PopList control is closed.

Syntax

```
void ofsClosePopList(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, control ID 52 represents
//the Poplist control that is closed.
ofsClosePopList ( "POPLIST", 52, OFS_ENDMSG, 332 );
```

ofsCloseWindow

Adds the Window_Close property (with Enabled attribute) to the current message. The Window_Close property indicates the act of closing a window.

Syntax

```
void ofsCloseWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current

	<p>message or if it also ends the current message. The end of a message requires special processing.</p> <p>OFS_ADD: Add the property to the current message.</p> <p>OFS_ENDMSG: Add property to the current message and end the current message.</p>
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, control ID 179 (named
//"Find Material Transactions") represents
//the window that is closed.
ofsCloseWindow( "Find Material Transactions ", 179, OFS_ENDMSG, 216 );
```

ofsCollapseTreeItem

Adds the Event-Collapsed property of a Tree control to the current message. The Event_Collapsed property indicates a Tree control item being collapsed.

Syntax

```
void ofsCollapseTreeItem(const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	<p>This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing.</p> <p>OFS_ADD: Add the property to the current message.</p> <p>OFS_ENDMSG: Add property to the current message and end the current message.</p>
PropertyID	Oracle-designated ID for the property being added.
Value	The relative position of the Tree control item.

Example

```
//In this example, item 4 (named "FORD") in Tree control ID 73 is collapsed.
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 180, "0" );
ofsSetPropertyPoint( "TREE", 73, OFS_ADD, 185, 19, 50);
ofsSetPropertyByte( "TREE", 73, OFS_ENDMSG, 186, "16" );
ofsRemoveFocus( "TEXTFIELD", 69, OFS_ENDMSG, 174 );
ofsFocus( "TREE", 73, OFS_ENDMSG, 174 );
ofsCollapseTreeItem( "TREE", 73, OFS_ENDMSG, 490, "4" ); /*Item value = FORD*/
ofsSendRecv(1 );
```

ofsColorAdd

Adds the Color_Add property to the current message. The Color_Add property is applied to the initial Forms environment.

Syntax

```
void ofsColorAdd(const char *sHandlerName, int ControlID, int ActionType, int PropertyID,
const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The color being applied to the Forms application environment.

Example

```
//The initial set of Forms statements describes the
//initial Forms environment. The description is matched
//on the server side. In this example, a color is
//defined as part of the Forms environment.

ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "90290" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
:
ofsColorAdd( "RUNFORM", 1, OFS_ADD, 284, "16776960" );
:
ofsSetRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );
:
ofsFocus( "BUTTON", 58, OFS_ENDMSG, 174 );
ofsSendRecv(1 );
```

ofsConnectToSocket

Establishes a socket-mode connection to the Oracle Forms Server.

Syntax

```
void ofsConnectToSocket(const char *Hostname, int Port);
```

Parameters

Parameter	Description
-----------	-------------

Hostname	Host name or IP address of the Oracle Forms Server.
Port	Port number used to connect to the Forms server.

Example

```
//In socket-mode, QALoad uses the IP address
//or host name of the server machine and the
//Form Server Port to execute a socket connection
//with the server.

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

ofsDeActivateWindow

Adds the Window_Activate property (with Disabled attribute) to the current message. The Window_Activate (with Disabled attribute) property indicates the ending of a currently opened window.

Syntax

```
void ofsDeActivateWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//This example shows the window "WINDOW_DATABASETTEST"
//being terminated prior to the ending of the HTTP session.

ofsDeActivateWindow( "WINDOW_DATABASETTEST", 24, OFS_ENDMSG, 247 );
ofsFocus( "BUTTON", 52, OFS_ENDMSG, 174 );
ofsSendRecv(1 );
ofsHTTPDisconnect();
```

ofsDefineTreeNode

Adds the Node_ID property of a Tree control to the current message. Node_ID property defines the relative position of the tree item, counting nested tree items.

Syntax

```
void ofsDefineTreeNode(const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *Value);
```


Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The relative position of the tree item, counting nested tree items.

Example

```
//In this example, the relative positions of items
//6 and 12 in Tree control ID 73 are defined.

ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "2" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "3" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "12" ); /*Item value = CLARK*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "0" );
ofsSendRecv(1 );
```

ofsDefineTreeNodeOffset

Adds the Node_Offset property of a Tree control to the current message. Node_Offset defines the relative position of the tree item, excluding nested tree items.

Syntax

```
void ofsDefineTreeNodeOffset(const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message.

	OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The relative position of the tree item, counting nested tree items .

Example

```
//In this example, the relative positions of
//items 6 and 12 in Tree control ID 73 are defined.

ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "2" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "3" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "12" ); /*Item value = CLARK*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "0" );
ofsSendRecv(1 );
```

ofsDeIconifyWindow

Adds the Window_Iconified property (with Disabled attribute) to the current message. This statement indicates a window being sized up from its icon representation.

Syntax

```
void ofsDeIconifyWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, window control ID 118 is
//being sized up from its icon representation.

ofsDeIconifyWindow ( "FORMWINDOW", 118, OFS_ENDMSG, 243 );
```

ofsDeSelectItem

Adds the Value property (with Disabled attribute) to the current Message. The Value property is applied to a Radio button, Checkbox, List Box or Combo Box control. This statement indicates the mouse moving away from a previously selected item that is associated with a Radio button, Checkbox, List Box or a Combo Box.

Syntax

```
void ofsDeSelectItem( const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, the mouse is deselecting Checkbox
//control ID 60 and selecting Radiobutton control ID 63.

ofsDeSelectItem( "CHECKBOX", 60, OFS_ENDMSG, 131 );
ofsSendRecv(1 );

ofsRemoveFocus( "CHECKBOX", 60, OFS_ENDMSG, 174 );
ofsFocus( "RADIOBUTTON", 63, OFS_ENDMSG, 174 );
ofsSendRecv(1 );

ofsSelectItem( "RADIOBUTTON", 63, OFS_ENDMSG, 131 );
ofsSendRecv(1 );
```

ofsDeselectTreeEvent

Adds the Event_DeSelect property of a Tree control to the current message. This statement indicates the application is moving from an internal processing event that is associated with a tree item.

Syntax

```
void ofsDeselectTreeEvent( const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.

ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The relative position of the tree control item.

Example

```
//In this example, user activity is moving
//away from Tree control ID 73.
ofsDeselectTreeEvent ( "TREE", 73, OFS_ENDMSG, 492, "1" );
```

ofsEdit

Adds the Value property to the current message. The property is associated with a Text Field control. This statement indicates the act of entering values into a text field.

Syntax

```
void ofsEdit( const char *sHandlerName, int ControlID, int ActionType, int PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The value entered in the text field.

Example

```
//In this example, the user enters "MFG"
//into Text Field control 75. The other
//statements are describing the location
//of the TextField control, the position
//of the cursor within the Text Field control,
//and recognizing the entry as a keyed entry.
```

```
ofsEdit( "TEXTFIELD", 75, OFS_ADD, 131, "MFG" );
ofsSetSelection( "TEXTFIELD", 75, OFS_ADD, 195, 3, 3);
ofsSetCursorPosition( "TEXTFIELD", 75, OFS_ENDMSG, 193, "3" );
ofsIndexSKey( "TEXTFIELD", 75, OFS_ENDMSG, 176, 9, 0);
ofsSendRecv(1 );
```

ofsExpandTreeItem

Adds the Event_Expanded property of a Tree control to the current message. The Event_Expanded property indicates a Tree control item being expanded.

Syntax

```
void ofsExpandTreeItem( const char *sHandlerName, int ControlID, int ActionType, int
PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The relative position of the tree item, counting nested tree items.

Example

```
//In this example, the item in Tree control
//ID 73 (named "CLARK") is expanded.

ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 180, "0" );
ofsSetPropertyPoint( "TREE", 73, OFS_ADD, 185, 12, 220);
ofsSetPropertyByte( "TREE", 73, OFS_ENDMSG, 186, "16" );
ofsExpandTreeItem( "TREE", 73, OFS_ENDMSG, 489, "12" ); /*Item value = CLARK*/
ofsSendRecv(1 );
```

ofsFindLOVValue

Adds the LOV_Find_Value property of a List of Values control to the current message. The statement indicates the user is searching for an item in a List of Values control. The search typically returns an item ID when a valid item is found for the given search string.

Syntax

```
void ofsFindLOVValue( const char *sHandlerName, int ControlID, int ActionType, int
PropertyID, const char *Value);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
Value	The name used to search the List of Values.

Example

```
//In this example, the user is using
//"CGI" string to search inside LOV
//control 85 (named "LISTVALUESDIALOG").
ofsFindLOVValue ( "LISTVALUESDIALOG", 85, OFS_ENDMSG, 454, "CGI" );
```

ofsFocus

Adds the Focus property (with Enabled attribute) to the current message. The Focus property typically indicates the mouse hovering on a GUI control.

Syntax

```
void ofsFocus(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, control ID 78 (a button)
//is the object of Focus. The button is
//subsequently clicked.

ofsFocus( "BUTTON", 78, OFS_ENDMSG, 174 );
ofsSendRecv(1 );
ofsClickButton( "BUTTON", 78, OFS_ENDMSG, 325 );
ofsSendRecv(1 );
```

ofsHideWindow

Adds the Visible property (with Disabled attribute) to the current message. The property is associated with a Window control. The statement indicates a window being hidden from view.

Syntax

```
void ofsHideWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, window control ID 118
//is being hidden from view.

ofsHideWindow( "FORMWINDOW", 118, OFS_ENDMSG, 173 );
```

ofsHTTPConnectToFormsServlet

Opens an HTTP connection to the Forms servlet responsible for initiating a Forms applet instance.

Syntax

```
void ofsHTTPConnectToFormsServlet(const char *sformsServletURL);
```

Parameters

Parameter	Description
sformsServletURL	The URL location of the Forms Servlet.

Example

```
ofsHTTPConnectToFormsServlet( "http://ntsap45b:7779/forms90/f90ervlet?ifcmd=startsession"
);
```

ofsHTTPConnectToListenerServlet

Opens an HTTP connection to the Forms Listener servlet responsible for starting an instance of the Forms run time process.

Syntax

```
void ofsHTTPConnectToListenerServlet(const char *sFormsServletURL);
```

Parameters

Parameter	Description
sformsServletURL	The URL location of the Forms Listener servlet.

Example

```
ofsHTTPConnectToListenerServlet( "http://ntsap45b:7779/forms90/l90ervlet");
```

ofsHTTPDisconnect

Closes the current HTTP connection to the Forms Listener Servlet.

Syntax

```
void ofsHTTPDisconnect();
```

Parameters

none

Example

```
ofsHTTPDisconnect();
```

ofsHTTPDoSSLHandshake

Establishes an SSL socket connection and starts an SSL handshake.

Syntax

```
void ofsHTTPDoSSLHandshake();
```

Parameters

none

Example

```
ofsHTTPDoSSLHandshake();
```

ofsHTTPSetHdrProperty

Establishes the HTTP headers to use for connecting to the Forms servlet and listener servlet.

Syntax

```
void ofsHTTPSetHdrProperty(const char *sName, const char *sValue);
```


Parameters

Parameter	Description
sName	The HTTP header name.
sValue	The HTTP header value.

Example

```
ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.9" );
ofsHTTPSetHdrProperty("Host", "ntsap45b.prod.ti.compuware.com:4445" );
ofsHTTPSetHdrProperty("Accept", "text/html, image/gif, image/jpeg, *; q=.2, "
    "**/*; q=.2" );
ofsHTTPSetHdrProperty("Connection", "Keep-alive" );
```

ofsHTTPSetListenerServletParms

Sets the Forms Listener Servlet parameters prior to connection.

Syntax

```
void ofsHTTPSetListenerServletParms(const char *sListenerServlet);
```

Parameters

Parameter	Description
sListenerServlet	Servlet parameters to use for this session.

Example

```
ofsHTTPSetListenerServletParms( "?ifcmd=getinfo&ifhost=C104444D01&ifip="
    "192.168.234.1" );
```

ofsHTTPInitialFormsConnect

Opens an HTTP connection to the Forms Listener servlet and posts the initial Forms handshake information.

Syntax

```
void ofsHTTPInitialFormsConnect();
```

Parameters

none

Example

```
ofsHTTPInitialFormsConnect();
```

ofsIconifyWindow

Adds the Window_Iconified property (with Enabled attribute) to the current message. This statement indicates a window being sized down to its icon representation.

Syntax

```
void ofsIconifyWindow( const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, window control ID 118 is
//being sized down to an icon.

ofsIconifyWindow ("FORMWINDOW", 118, OFS_ENDMSG, 243);
```

ofsIndexKey

Adds the Index_Key property to the current message. The Index_Key property typically indicates a keyed entry in a TextField control, such as a user ID entry.

Syntax

```
void ofsIndexKey(const char *sHandlerName, int iControlId, int iAction, int iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCooriantex	X coordinate of the keyed entry.

iCoordinateY

Y coordinate of the keyed entry.

Example

```
//In this example, the user enters "M" into Text Field control 75.
//The other statements are describing the location of the TextField control,
//the position of the cursor within the Text Field control,
//and recognizing the entry as a keyed entry.

ofsEdit( "TEXTFIELD", 75, OFS_ADD, 131, "M" );
ofsSetSelection( "TEXTFIELD", 75, OFS_ADD, 195, 1, 1);
ofsSetCursorPosition( "TEXTFIELD", 75, OFS_ENDMSG, 193, "1" );
ofsIndexKey( "TEXTFIELD", 75, OFS_ENDMSG, 175, 97, 0);
ofsSendRecv(1 );
```

ofsIndexSKey

Adds the `Index_SKey` property to the current message. The `Index_SKey` property is typically associated with a keyed entry in a `TextField` control, such as a user ID entry.

Syntax

```
void ofsIndexSKey(const char *sHandlerName, int iControlID, int iAction, int iPropertyID,
int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate of the keyed entry.
iCoordinateY	Y coordinate of the keyed entry.

Example

```
//In this example, the user enters "MFG" into Text Field control 75.
//The other statements are describing the location of the TextField control,
//the position of the cursor within the Text Field control,
//and recognizing the entry as a keyed entry.

ofsEdit( "TEXTFIELD", 75, OFS_ADD, 131, "MFG" );
ofsSetSelection( "TEXTFIELD", 75, OFS_ADD, 195, 3, 3);
ofsSetCursorPosition( "TEXTFIELD", 75, OFS_ENDMSG, 193, "3" );
ofsIndexSKey( "TEXTFIELD", 75, OFS_ENDMSG, 176, 9, 0);
ofsSendRecv(1);
```

ofsInitSessionCmdLine

Adds the INITIAL CMDLINE property to the current message. The INITIAL CMDLINE property is applied to the initial Forms environment.

Syntax

```
void ofsInitSessionCmdLine(const char *sClassName, int iHandlerID, int iAction, int iPropertyID, const char *sCmdLineInfo);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sCmdLineInfo	Value of the Initial CmdLine property.

Example

```
ofsInitSessionCmdLine("RUNFORM", 1, OFS_ADD, 265,
    "server module=/oracle/appl/vis11iappl/fnd/11.5.0/forms/US/FNDSCSGN userid=APPLS"
    "YSPUB/PUB@vis11i fndnam=APPS");
```

ofsInitSessionTimeZone

Adds the Time_Zone property to the current message. The Time_Zone property is applied to the initial Forms environment.

Syntax

```
void ofsInitSessionTimeZone(const char *sClassName, int iHandlerID, int iAction, int iPropertyID, const char *sTimeZone);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message.

	OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sTimeZone	The time zone to use for this session. The time zone is specified by the application.

Example

```
ofsInitSessionTimeZone ( "RUNFORM", 1, OFS_ENDMSG, 530, "America/New_York" );
```

ofsListItemValue

Adds the List_Item property of a PopList or a TList control to the current message. This statement defines an item in a PopList or a TList control.

Syntax

```
void ofsListItemValue(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Relative position of the item in the PopList or TList control.

Example

```
//In this example, item 6 is defined in Poplist control ID 66.  
//Item 6 is labeled "Cindy Wang."  
ofsListItemValue( "POPLIST", 66, OFS_ENDMSG, 131, "6" ); /* Item value = Cindy Wang*/
```

ofsLoadValue

Loads the values of a byte array or a string array associated with a GUI control. This statement only applies when the size of the byte array or string array > 0.

Syntax

```
void ofsLoadValue(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value of the item in the byte array or string array associated with a GUI control.

Example

```
//In this example, value 7 is being added to the list of values in the array.
ofsLoadValue( "RUNFORM", 1, OFS_ENDMSG, 1, "7");
```

ofsLOVRequestRow

Adds the LOV_REQUEST_ROW property to the current message. This statement defines an item in a List of Values control.

Syntax

```
void ofsLOVRequestRow(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, int iPosX, int iPosY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iPosX	X coordinate of the item in the LOV control.

iPosY	Y coordinate of the item in the LOV control.
-------	--

Example

```
//In this example, the position of an item in LOV control 85 is defined.
ofsLOVRequestRow( "LISTVALUESDIALOG", 85, OFS_ENDMSG, 451, 5, 1);
```

ofsLOVSelection

Adds the LOV_SELECTION property to the current message. This statement indicates an item being selected from a List of Values.

Syntax

```
void ofsLOVSelection(const char *sHandlerName, int iControlID, int iAction, int iPropertyID,
const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Relative position of the selected item in the List of Values control.

Example

```
//In this example, item 1 from LOV control ID 264 is selected
ofsActivateWindow( "NAVIGATOR", 28, OFS_ENDMSG, 247 );
ofsLOVSelection( "LISTVALUESDIALOG", 264, OFS_ENDMSG, 450, "1" );
ofsSendRecv(1);
```

ofsMenuParamDlgOK

Adds the MENUPARAM_DLGOK property to the current message. This statement defines the text in the menu param dialog control.

Syntax

```
void ofsMenuParamDlgOK(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
-----------	-------------

sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The text value of the menu param dialog.

Example

```
//In this example, dialog control ID 12 has a text title of "testButton".
OfsMenuParamDlgOK( "menu1", 12, OFS_ENDMSG, 16, "testbutton");
```

ofsOpenWindow

Adds the Window_Close property (with Disabled attribute) to the current message. The statement indicates the act of opening a window.

Syntax

```
void ofsOpenWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, control ID 52 (named
//"Sample Window") represents the window
//that is opened.
ofsOpenWindow ( "Sample Window", 52, OFS_ENDMSG, 216 );
```


ofsRemoveFocus

Adds the Focus property (with Disabled attribute) to the current message. The RemoveFocus property typically indicates the mouse moving away from a GUI control.

Syntax

```
void ofsRemoveFocus(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, Focus is moved from control
//ID 77 (a Text Field) to control ID 78 (a button).

ofsRemoveFocus( "TEXTFIELD", 77, OFS_ENDMSG, 174 );
ofsFocus( "BUTTON", 78, OFS_ENDMSG, 174 );
ofsSendRecv(1 );
```

ofsScroll

Adds the Block_Scroller property to the current message. This statement indicates a scrolling activity.

Syntax

```
void ofsScroll(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current

	message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The value associated with the scroll bar

Example

```
ofsScroll( "RUNFORM", 1, OFS_ADD, 250, "2");
```

ofsScrollSize

Adds the Block_Scroller_Size property to the current message. This statement indicates the block scroller size.

Syntax

```
ofsScrollSize(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The value associated with the scroll bar

Example

```
ofsScrollSize( "RUNFORM", 1, OFS_ADD, 256, "12");
```

ofsSelectItem

Adds the Value property (with Enabled attribute) to the current Message. The Value property is applied to a Radio button, Checkbox, List Box or Combo Box control. This statement indicates an item associated with a Radio button, Checkbox, List Box or a Combo Box is being selected.

Syntax

```
void ofsSelectItem(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
-----------	-------------

sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, the mouse is deselecting Checkbox
//control ID 60 and selecting Radiobutton control ID 63.

ofsDeselectItem( "CHECKBOX", 60, OFS_ENDMSG, 131 );
ofsSendRecv(1 );

ofsRemoveFocus( "CHECKBOX", 60, OFS_ENDMSG, 174 );
ofsFocus( "RADIOBUTTON", 63, OFS_ENDMSG, 174 );
ofsSendRecv(1 );

ofsSelectItem( "RADIOBUTTON", 63, OFS_ENDMSG, 131 );
ofsSendRecv(1 );
```

ofsSelectMenuItem

Adds the Menu_Event property to the current message. This statement indicates an item being selected from the Forms Event Menu.

Syntax

```
void ofsSelectMenuItem(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the menu item.

Example

```
//In this example, a menu item valued 3 is selected. The menu item
//is associated with control ID 1 (Runform).
ofsSelectMenuItem( "RUNFORM", 1 , OFS_ADD, 477, "3");
```

ofsSelectTreeEvent

Adds the Selected_Event property of a Tree Control to the current message. This statement indicates a Tree item being selected. The selected item is associated with an internal processing event.

Syntax

```
void ofsSelectTreeEvent(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.
sValue	Relative position of the selected Tree item.

Example

```
//In this example, the user selects item 2 of Tree control ID 15.
//Item 2 has an internal processing event.
ofsSelectTreeEvent( "TREE", 15, OFS_ADD, 488, 2);
```

ofsSendRecv

Sends the client request as Forms messages to the Forms server, gets the server response, and reads the responses as Forms messages.

Syntax

```
void ofsSendRecv(int iResponseCode);
```

Parameters

Parameter	Description
iResponseCode	The response code associated with the client request's terminal message. (1= add, 2=update, 3=close).

Example

```
//In this example, the messages sent to the server include a Text Field
//location attribute and a Window size attribute.

ofsSetSelection( "TEXTFIELD", 75, OFS_ENDMSG, 195, 0, 0);
ofsSetWindowSize( "FORMWINDOW", 6, OFS_ENDMSG, 137, 1024, 768);
ofsSendRecv(1 );
```

ofsServerSideDisconnect

Disconnects QALoad's socket connection to the server-side code. The server-side code intercepts the messages between QALoad and the Forms Listener servlet.

Syntax

```
void ofsServerSideDisconnect();
```

Parameters

none

Example

```
ofsServerSideDisconnect();
```

ofsSetColorDepth

Adds the Color_Depth property to the current message. The Color_Depth property is applied to the initial Forms environment.

Syntax

```
void ofsSetColorDepth(const char *sClassName, int iHandlerID, int iAction, int iPropertyID,
const char *sColorDepth);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sColorDepth	Value associated with the color depth.

Example

```
ofsSetColorDepth( "RUNFORM", 1, OFS_ADD, 266, "256" );
```

ofsSetCursorPosition

Adds the `Cursor_Position` property of a Text Field control to the current message. The `Cursor_Position` property indicates the relative position of the cursor in the Text Field control at the time of user entry.

Syntax

```
void ofsSetCursorPosition(const char *sHandlerName, int iControlId, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
<code>sHandlerName</code>	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
<code>iControlID</code>	Captured ID of the GUI control for the current message.
<code>iAction</code>	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
<code>iPropertyID</code>	Oracle-designated ID for the property being added.
<code>sValue</code>	Relative position of the cursor in the Text Field control at the time of user entry.

Example

```
//In this example, the cursor is positioned on the 7th character.  
ofsSetCursorPosition( "TEXTFIELD", 77, OFS_ENDMSG, 193, "7" );
```

ofsSetDisplaySize

Adds the `Display_Size` property to the current message. The `Display_Size` property is applied to the initial Forms environment.

Syntax

```
void ofsSetDisplaySize(const char *sClassName, int iHandlerID, int iAction, int iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
<code>sClassName</code>	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
<code>iHandlerID</code>	Captured ID of the GUI control for the current message.
<code>iAction</code>	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing.

	OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate of the canvas display.
iCoordinateY	Y coordinate of the canvas display.

Example

```
ofsSetDisplaySize( "RUNFORM", 1, OFS_ADD, 264, 1024, 768);
```

ofsSetErrorDialogTitle

Adds the DISPLAYERRORDIALOG_TITLE property to the current message. This statement defines the text title associated with the Display Error Dialog control.

Syntax

```
void ofsSetErrorDialogTitle(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Text title associated with the Error Dialog control.

Example

```
//In this example, the text title of error dialog control ID 12 is defined as "TestErr1".  
ofsSetErrorDialogTitle( "ERRDLG1", 12, OFS_ADD, 129, "TestErr1");
```

ofsSetExpectedServerMsg

Enables the script to continue if a known error or warning message is received from the server. It is positioned before the ofsSendRecv statement, which checks the server reply messages. If error message checking is enabled, and the server message contains "FRM-", "ORA-" or "APP-", ofsSendRecv throws an exception unless it is preceded by ofsSetExpectedServerMsg.

Syntax

```
void ofsSetExpectedServerMsg(const char *ExpectedServerMessage);
```

Parameters

Parameter	Description
ExpectedServerMessage	The expected server message.

Example

```
//Before sending the request to the server with the statement ofsSendRecv,
//QALoad stores the expected message from the server reply,
//so that Playback would ignore the server message and continue execution.
.
.
.
ofsSelectMenuItem( "WINDOW_START_APP", 11, OFS_ENDMSG, 477, "MENU_77" );
ofsSetExpectedServerMsg("FRM-41003: This function cannot be performed here.");
ofsSendRecv(1 );
```

ofsSetFontName

Adds the Font_Name property to the current message. The Font_Name property is applied to the initial Forms environment.

Syntax

```
void ofsSetFontName(const char *sClassName, int iHandlerID, int iAction, int iPropertyID,
const char *sFontName);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sFontName	Name of the font to use.

Example

```
ofsSetFontName( "RUNFORM", 1, OFS_ADD, 383, "Dialog" );
```


ofsSetFontSize

Adds the Font_Name property to the current message. The Font_Name property is applied to the initial Forms environment.

Syntax

```
void ofsSetFontName(const char *sClassName, int iHandlerID, int iAction, int iPropertyID,
const char *sFontName);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sFontName	The size of the font to use.

Example

```
ofsSetFontSize( "RUNFORM", 1, OFS_ADD, 377, "900" );
```

ofsSetFontStyle

Adds the Font_Style property to the current message. The Font_Style property is applied to the initial Forms environment.

Syntax

```
void ofsSetFontStyle(const char *sHandlerName, int iControlId, int iAction, int iPropertyID,
const char *sValue);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current

	m message.
iPropertyID	Oracle-designated ID for the property being added.
sFontName	The style of the font to use.

Example

```
ofsSetFontStyle( "RUNFORM", 1, OFS_ADD, 378, "0" );
```

ofsSetFontWeight

Adds the Font_Weight property to the current message. The Font_Weight property is applied to the initial Forms environment.

Syntax

```
void ofsSetFontWeight(const char *sHandlerName, int iControlId, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sFontName	The font weight to use.

Example

```
ofsSetFontWeight( "RUNFORM", 1, OFS_ADD, 379, "0" );
```

ofsSetICXTicket

Sets the value of the ICX ticket for the current Oracle Applications login. The statement is used only in a Universal OFS-WWW session, as a replacement for the OracleAppsLogin statement. The statement is manually added to the script, along with the WWW statement DO_GetUniqueString. For more information, see OFS Advanced Scripting Techniques.

Syntax

```
ofsSetICXTicket( char **cookieValue);
```

Parameters

Parameter	Description
-----------	-------------

cookieValue

Address to a string where the cookie value is stored.

Example

```

...
/* Declare Variables */
...
char *p;
char ICX_Ticket[100];
char *pTicket;
...
...
BEGIN_TRANSACTION();
...
...
// This statement should be added after the request line that returns the ICX ticket
p = DO_GetUniqueString( "icx_ticket=", "" );
strcpy( ICX_Ticket, p );
pTicket=ICX_Ticket;

// Verify the ICX ticket value
RR__printf("ICX_Ticket=\"%s\"\n", ICX_Ticket);

// The ofsSetICXTicket statement passes the ICX ticket value to the
ofsInitiSessionCmdLine statement
ofsSetICXTicket(&pTicket);

```

ofsSetInitialVersion

Adds the Initial_Version property to the current message. The Initial_Version property is applied to the initial Forms environment.

Syntax

```
void ofsSetInitialVersion(const char *sClassName, int iHandlerID, int iAction, int
iPropertyID, const char *sFormsVersion);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sFormsVersion	The Forms Version of the captured application.

Example

```
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "60818" );
```

ofsSetJavaContainerArgName

Adds the JAVACONTAINER_ARG_NAME property to the current message. This statement defines the name assigned to an item in a JavaContainer control.

Syntax

```
void ofsSetJavaContainerArgName(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Name assigned to a JavaContainer control item.

Example

```
ofsSetJavaContainerArgName("Test_App", 15, OFS_ADD, 400, "TestBeanItem");
```

ofsSetJavaContainerArgValue

Adds the JAVACONTAINER_ARG_VALUE property to the current message. This statement defines the value entered by the user in a JavaContainer control item.

Syntax

```
void ofsSetJavaContainerArgValue(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message

	requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The value entered by the user in a JavaContainer control item.

Example

```
ofsSetJavaContainerArgValue("Test_App", 15, OFS_ADD, 401, "BeanEntry1");
```

ofsSetJavaContainerEvent

Adds the JAVACONTAINER_ARG_EVENT property to the current message. This statement defines the name assigned to a JavaContainer control.

Syntax

```
void ofsSetJavaContainerEvent(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Name assigned to the JavaContainer control.

Example

```
ofsSetJavaContainerEvent("Test_App", 15, OFS_ADD, 399, "TestEvent1");
```

ofsSetLogonDatabase

Adds the LOGON_DATABASE property to the current message. This statement defines the connect string entry in the Forms Logon dialog.

Syntax

```
void ofsSetLogonDatabase(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The logon database for this session.

Example

```
ofsSetLogonDatabase( "Logon", 34, OFS_ENDMSG, 435, "iasdb" );
```

ofsSetLogonPassWord

Adds the LOGON_PASSWORD property to the current message. This statement defines the password entry in the Forms Logon dialog.

Syntax

```
void ofsSetLogonPassWord(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The password for this session.

Example

```
ofsSetLogonPassWord( "Logon", 34, OFS_ADD, 434, "tiger" );
```

ofsSetLogonUserName

Adds the LOGON_USERNAME property to the current message. This statement defines the user name entry in the Forms Logon dialog.

Syntax

```
void ofsSetLogonUserName(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	The user name to use for this session.

Example

```
ofsSetLogonUserName( "Logon", 34, OFS_ADD, 433, "scott" );
```

ofsSetNoRequiredVAList

Adds the Required_VA_List property (with Disabled attribute) to the current message. The Required_VA_List property is applied to the initial Forms environment.

Syntax

```
void ofsSetNoRequiredVAList(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current

	message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//The initial set of Forms statements describes
//the initial Forms environment. The description
//is matched on the server side.

ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "60818" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);

.
.
.

ofsSetNoRequiredVList( "RUNFORM", 1, OFS_ADD, 291 );

.
.
.

ofsInitSessionTimeZone( "RUNFORM", 1, OFS_ENDMSG, 527, "EST" );
ofsSendRecv(1 );
```

ofsSetPropertyBoolean

Adds the generic boolean property (with Enabled attribute) to the current message. Use this statement when the boolean property is not known to QALoad.

Syntax

```
void ofsSetPropertyBoolean(const char *sHandlerName, int iControlID, int iAction, int iPropertyID);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, property 381 is of Boolean type, and is associated with Tree control ID 73.

ofsSetPropertyBoolean("TREE", 73, OFS_ENDMSG, 381);
```


ofsSetPropertyByte

Adds the generic byte property to the current message. This statement is used when the byte property is not known to QALoad.

Syntax

```
void ofsSetPropertyByte(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the byte property.

Example

```
//In this example, property 186 is of type Byte, and is associated with Tree control ID 73.  
ofsSetPropertyByte( "TREE", 73, OFS_ENDMSG, 186, "16" );
```

ofsSetPropertyByteArray

Adds the generic byte array property to the current message. This statement is used when the byte array property is not known to QALoad.

Syntax

```
void ofsSetPropertyByteArray(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message.

	OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Size of the byte array.

Example

//In this example, property 382 is of Byte Array type, and is associated with Tree control ID 73.

```
ofsSetPropertyByteArray( "TREE", 73, OFS_ENDMSG, 382, "0" );
```

ofsSetPropertyCharacter

Adds the generic Character property to the current message. This statement is used when the Character property is not known to QALoad.

Syntax

```
void ofsSetPropertyCharacter(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the Character property.

Example

//In this example, property 383 is of Character type, and is associated with Tree control ID 73.

```
ofsSetPropertyCharacter( "TREE", 73, OFS_ADD, 383, "20");
```

ofsSetPropertyDate

Adds the generic Date property to the current message. This statement is used when the Date property is not known to QALoad.

Syntax

```
void ofsSetPropertyDate(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the Date property.

Example

```
//In this example, property 383 is of Date type, and is associated with Tree control ID 73.
ofsSetPropertyDate( "TREE", 73, OFS_ADD, 383, "2000-02-22");
```

ofsSetPropertyFloat

Adds the generic Float property to the current message. This statement is used when the Float property is not known to QALoad.

Syntax

```
void ofsSetPropertyFloat(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the Float property.

Example

```
//In this example, property 383 is of Float type, and is associated with Tree Control ID 73.
ofsSetPropertyFloat( "TREE", 73, OFS_ADD, 383, "2000.0222");
```

ofsSetPropertyInteger

Adds the generic Integer property to the current message. This statement is used when the Integer property is not known to QALoad.

Syntax

```
void ofsSetPropertyInteger(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the Integer property.

Example

```
//In this example, property 383 is of Integer type, and is associated with Tree control ID 73.
ofsSetPropertyInteger( "TREE", 73, OFS_ADD, 383, "20");
```

ofsSetPropertyPoint

Adds the generic Point property to the current message. This statement is used when the Point property is not known to QALoad.

Syntax

```
void ofsSetPropertyPoint(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.

iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate value of the Point property.
iCoordinateY	Y coordinate value of the Point property.

Example

//In this example, property 185 is of Point type, and is associated with Tree control ID 73.
ofsSetPropertyPoint("TREE", 73, OFS_ADD, 185, 30, 50);

ofsSetPropertyRectangle

Adds the generic Rectangle property to the current message. This statement is used when the Rectangle property is not known to QALoad.

Syntax

```
void ofsSetPropertyRectangle(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, int iXval, int iYval, int iWval, int iHval);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iXval	X value of the rectangle.
iYval	Y value of the rectangle.
iWval	Width of the rectangle.
iHval	Height value of the rectangle.

Example

```
//In this example, property 155 is of type Rectangle, and is associated with Control ID 73
//(named "Button").
ofsSetPropertyRectangle( "BUTTON", 73, OFS_ADD, 155, 0, 0, 106, 29);
```

ofsSetPropertyString

Adds the generic **String** property to the current message. This statement is used when the **String** property is not known to QALoad.

Syntax

```
void ofsSetPropertyString(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the String property.

Example

```
//In this example, property 520 is of String type, and is associated with control ID 1
(RunForm).
ofsSetPropertyString( "RUNFORM",1, OFS_ENDMSG, 530, "America/New_York" );
```

ofsSetPropertyStringArray

Adds the generic **String** array property to the current message. This statement is used when the **String** array property is not known to QALoad.

Syntax

```
void ofsSetPropertyStringArray(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.

iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Size of the String array.

Example

```
//In this example, property 382 with size 0 is of type String Array.  
//The property is associated with Tree control ID 73.
```

```
ofsSetPropertyStringArray( "TREE", 73, OFS_ENDMSG, 382, "0" );
```

ofsSetPropertyVoid

Adds the generic Void property to the current message. This statement is used when the Void property is not known to QALoad.

Syntax

```
void ofsSetPropertyVoid(const char *sHandlerName, int iControlID, int iAction, int iPropertyID);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, property 382 is of Void type and is associated with Tree control ID 73.
```

```
ofsSetPropertyVoid( "TREE", 73, OFS_ENDMSG, 382 );
```

ofsSetRequiredVAList

Adds the Required_VA_List property (with Enabled attribute) to the current message. The Required_VA_List property is applied to the initial Forms environment.

Syntax

```
void ofsSetRequiredVAList(const char *HandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//The initial set of Forms statements describes
//the initial Forms environment. The description
//is matched on the server side.

ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "90290" );
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);

.
.
.

ofsSetRequiredVAList( "RUNFORM", 1, OFS_ADD, 291 );

.
.
.

ofsFocus( "BUTTON", 58, OFS_ENDMSG, 174 );
ofsSendRecv(1 );
```

ofsSetRunOptions

Sets the runtime values for CONNECT TYPE, HEARTBEAT, and CHECK SERVER MESSAGES

Syntax

```
void ofsSetRunOptions(const char *sFormsVersion, int iConnectType, int iHeartbeatInterval, int iCheckServerMsgs);
```

Parameters

Parameter	Description
-----------	-------------

sFormsVersion	Version of Oracle Forms in use.
iConnectType	This flag indicates the type of connection to establish with the server. OFS_SOCKET: Socket connection OFS_HTTP: HTTP connection OFS_HTTPS: Secure (SSL) connection
iHeartbeatInterval	Heartbeat interval (seconds).
iCheckServerMsgs	This flag indicates whether QALoad checks server messages. If server message checking is enabled, the script fails if the server sends a message ("FRM-", "ORA-", "APP-") unless the message is set as an expected message (see ofsSetExpectedServerMsg()). In addition, ofsSetServerFailedMsg() overrides the expected message if the message matches the failed message. OFS_DONTCHECKMSGS: Do not fail when the server sends messages back OFS_CHECKMSGS: Fail on receipt of messages from server

Example

```
ofsSetRunOptions( "6i", OFS_SOCKET, 4, OFS_CHECKMSGS );
```

ofsSetScaleInfo

Adds the Scale property to the current message. The Scale property is applied to the initial Forms environment.

Syntax

```
void ofsSetScaleInfo(const char *sClassName, int iHandlerID, int iAction, int iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate associated with scale property.
iCoordinateY	Y coordinate associated with scale property.

Example

```
ofsSetScaleInfo( "RUNFORM", 1, OFS_ADD, 267, 11, 18);
```

ofsSetScreenResolution

Adds the Screen Resolution property to the current message. The Screen Resolution property is applied to the initial Forms environment.

Syntax

```
void ofsSetScreenResolution(const char *sClassName, int iHandlerID, int iAction, int iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
sClassName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate associated with the property.
iCoordinateY	Y coordinate associated with the property.

Example

```
ofsSetScreenResolution( "RUNFORM", 1, OFS_ADD, 263, 96, 96);
```

ofsSetSelection

Adds the Selection property of a Text Field control to the current message. This statement indicates the selected Text Field location during user entry.

Syntax

```
void ofsSetSelection(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, int iCoordinateX, int iCoordinateY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message.

	OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iCoordinateX	X coordinate of the text field entry.
iCoordinateY	Y coordinate of the text field entry.

Example

```
ofsSetSelection( "TEXTFIELD", 75, OFS_ADD, 195, 0, 0);
ofsSetCursorPosition( "TEXTFIELD", 75, OFS_ENDMSG, 193, "0" );
ofsSetWindowLocation( "Oracle Applications", 32, OFS_ENDMSG, 135, 231, 218);
ofsShowWindow( "Oracle Applications", 32, OFS_ENDMSG, 173 );
ofsActivateWindow( "Oracle Applications", 32, OFS_ENDMSG, 247 );
ofsFocus( "TEXTFIELD", 75, OFS_ENDMSG, 174 );
ofsSendRecv(2 );
```

ofsSetServerFailedMsg

Enables QALoad to fail playback based on the user-entered string and filter parameters. This statement overrides the effects of the ofsSetExpectedMsg statement, which enables QALoad to continue playback if FRM-, ORA- or APP- server messages are encountered.

Syntax

```
void ofsSetServerFailedMsg(const char *sMsgString, int iMsgOption);
```

Parameters

Parameter	Description
sMsgString	String to compare against server message
iMsgOption	This flag indicates the string comparison method to use for comparing a string against a server message. All of these methods are case sensitive. OFS_KEYWORD: Search for string anywhere in server message. OFS_PREFIX: Compare string against beginning of the message. OFS_SUFFIX: Compare string against end of the message. OFS_ENTIRE_MSG: Compare string against entire message.

Example

```
ofsSetServerFailedMsg( "FRM-4041", OFS_KEYWORD );
:
ofsSetExpectedMsg("FRM-4041").;
ofsSendRecv(1);
```

ofsSetServletMode

Creates a socket connection to the server-side code which communicates with the Forms Listener Servlet. The server-side code intercepts messages between QALoad and the servlet during a server-side connection.

Syntax

```
void ofsSetServletMode(int iConnectMode, const char *sServletName);
```

Parameters

Parameter	Description
iConnectMode	Connection mode.
sServletName	The listener servlet name.

Example

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

ofsSetValue

Adds a generic Value property to the current message.

Syntax

```
void ofsSetValue(const char *sHandlerName, int iControlID, int iAction, int iPropertyID,
const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the class name of the control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the property.

Example

```
//In this example, a value of "30" is being associated with control ID 1 "RUNFORM"  
ofsSetValue( "RUNFORM", 1, OFS_ADD, 131, "30");
```

ofsSetWindowLocation

Adds the Location property of a Window control to the current message. This statement defines the window location in the canvas.

Syntax

```
void ofsSetWindowLocation(const char *sHandlerName, int iControlID, int iAction, int
iPropertyID, int iPosX, int iPosY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.
iPosX	X coordinate of the window control.
iPosY	Y coordinate of the window control.

Example

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

ofsSetWindowSize

Adds the **Size** property of a Window control to the current message. This statement indicates a window being resized.

Syntax

```
void ofsSetWindowSize(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, int iPosX, int iPosY);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
iPropertyID	Oracle-designated ID for the property being added.

iPosX	Width of the window control.
iPosY	Length of the window control.

Example

```
ofsSetWindowSize( "FORMWINDOW", 6, ofs_ENDMSG, 137, 650, 500);
ofsSendRecv(1 );
```

ofsShowWindow

Adds the Visible property (with Enabled attribute) to the current message. The property is associated with a Window control. The statement indicates a window being displayed in front of all other windows.

Syntax

```
void ofsShowWindow(const char *sHandlerName, int ControlID, int ActionType, int PropertyID);
```

Parameters

Parameter	Description
HandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
ControlID	Captured ID of the GUI control for the current message.
ActionType	This flag indicates if the property is only to be added to the current message or if it also ends the current message. The end of a message requires special processing. OFS_ADD: Add the property to the current message. OFS_ENDMSG: Add property to the current message and end the current message.
PropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, window control ID 118 is
//being displayed in front of all other windows.
ofsShowWindow( "FORMWINDOW", 118, ofs_ENDMSG, 173 );
```

ofsSocketDisconnect

Closes the connection of a socket-mode playback.

Syntax

```
void ofsSocketDisconnect();
```

Parameters

none

Example

```
ofsSocketDisconnect();
```

ofsStartSubMessage

Adds a sub-message to the current message. A sub-message is a message nested inside another message.

Syntax

```
void ofsStartSubMessage(const char *sHandlerName, int iHandlerID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iHandlerID	Captured ID of the GUI control for the current message.
iAction	Action type. Adds the property of the succeeding nested message to the current message.
iPropertyID	Oracle-designated ID for the property being added.
sValue	Value associated with the parent message.

Example

```
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "2" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "6" ); /*Item value = BLAKE*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "3" );
ofsStartSubMessage( "TREE", 73, OFS_STARTSUBMSG, 505, "0" );
ofsDefineTreeNode( "TREE", 73, OFS_ADD, 500, "12" ); /*Item value = CLARK*/
ofsDefineTreeNodeOffset( "TREE", 73, OFS_ENDMSG, 503, "0" );

ofsSendRecv(1 );
```

ofsTabControlTopPage

Adds the TabControl_Top_Page property to the current message.

Syntax

```
void ofsTabControlTopPage(const char *sHandlerName, int iControlID, int iAction, int iPropertyID, const char *sValue);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	Action type. Adds the property of the succeeding nested message to the current message.
iPropertyID	Oracle-designated ID for the property being added.

sValue	Value associated with the Tab Control.
--------	--

Example

```
ofsTabControlTopPage( "RUNFORM", 1, OFS_ENDMSG, 411, "30");
```

ofsUnsetPropertyBoolean

Adds a generic Boolean property (with Disabled attribute) to the current message. This statement is used when the property is not known to QALoad.

Syntax

```
void ofsUnsetPropertyBoolean(const char *sHandlerName, int iControlID, int iAction, int iPropertyID);
```

Parameters

Parameter	Description
sHandlerName	Captured name of the control ID for the current message. When the control name is not available, the Class name of the Control ID is shown.
iControlID	Captured ID of the GUI control for the current message.
iAction	Action type. Adds the property of the succeeding nested message to the current message.
iPropertyID	Oracle-designated ID for the property being added.

Example

```
//In this example, property 382 is of Boolean type,
//and is associated with control ID 1 (Runform).
ofsUnsetPropertyBoolean( "RUNFORM", 1, OFS_ADD, 382);
```

OracleAppsLogin

This method simulates an Oracle Applications 11i login (Personal Home Page) and retrieves the icx_ticket associated with that login. This statement appears on the script when the Convert option for OracleAppsLogin is enabled. The URL parameter should be the actual Oracle Personal Home Page address. Exceptions are thrown if the page cannot be found or if the icx_ticket isn't returned by the server.

Syntax

```
OracleAppsLogin(const char *ServerURL, const char *UserID, const char *Password)
```

Parameters

Parameter	Description
ServerURL	URL of the Oracle Apps Login page (the Personal Home Page address).
UserID	User ID that was entered on the login page during the recording.
Password	Password that was entered on the login page during the recording.

Example

```
//QALoad posts the login information to the homepage URL, retrieves the
//ICX ticket from the reply and uses the ICX ticket in a subsequent Forms message.
OracleAppsLogin( http://appl11..com:8000/oraclemypage.home, "myuserid", "mypassword" );
```

OracleForms

The constructor for the OracleForms class. This should be classed on ce per script.

Syntax

```
OracleForms( QALoad BaseScript )
```

Parameters

Parameter	Description
QALoad BaseScript	Reference to the current script.

Exceptions

None thrown.

Example

```
OracleForms oracleForms = new OracleForms( this );
```

OracleFormsMsg

This method constructs a new OracleFormsMsg.

It should be used rather than calling the public constructor of the OracleFormsMsg class directly.

Syntax

```
OracleFormsMsg(actionCode, handlerClassId, handlerId )
OracleFormsMsg(actionCode, handlerClassId, controlName)
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
ActionCode int	Type of action this message refers to. This value should not be changed from that specified in the generated script.
HandlerClassId int	Type of control. Generally this value is 0 and should not be changed from that specified in the generated script.
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms

application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.OracleFormsMsg( 2, 0, TEST_BUTTON);
oracleFormsMsg1 = oracleForms.OracleFormsMsg( 2, 0, "BLOCK3.TEST_BUTTON_0" );
```

PopList

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
PopList( handlerID );
PopList( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.PopList( CONTROL_001 );
oracleFormsMsg1 = oracleForms.PopList( "CONTROLabc" );
```

PopupHelp

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
PopupHelp( handlerID );
PopupHelp( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.

ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.
--------------------	---

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.PopupHelp( CONTROL_001 );
oracleFormsMsg1 = oracleForms.PopupHelp( "CONTROLabc" );
```

PromptList

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
PromptList( handlerID );
PromptList( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.PromptList( CONTROL_001 );
oracleFormsMsg1 = oracleForms.PromptList( "CONTROLabc" );
```

RadioButton

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
RadioButton( handlerID );
RadioButton( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or

	type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.RadioButton( CONTROL_001 );
oracleFormsMsg1 = oracleForms.RadioButton( "CONTROLabc" );
```

Runform

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
Runform( handlerID );
Runform( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

QALoad Exception ("No match for control name: <control_name>");

Example

```
oracleFormsMsg1 = oracleForms.Runform( CONTROL_001 );
oracleFormsMsg1 = oracleForms.Runform( "CONTROLabc" );
```

SetExpectedServerMsg

This method allows the script to continue processing if a known error or warning message is received from the server.

The XmitTerminalMessage method, after it transmits the terminal message, checks the Oracle Forms Server for incoming data messages. If error message checking is enabled, and if any of these messages contain a warning or error message, XmitTerminalMessage will throw an exception.

However, if a message is received that contains the message text specified in this method, the error or warning message will be ignored.

Syntax

```
SetExpectedServerMsg( msgText )
```

Parameters

Parameter	Description
msgText String	The error message expected from the server.

Exceptions

None

Example

```
oracleForms.SetExpectedServerMsg( "FRM-4041" );
```

SetHeartbeat

Sets the interval time for the transmission of internal heartbeat messages to the server.

Without this statement, Playback takes the default —2 minutes. If the parameter is set to 0, transmission of heartbeat messages is suppressed.

Syntax

```
SetHeartbeat( minutes );
```

Parameters

Parameter	Description
minutes int	Minute-intervals for internal heartbeat messages.

Exceptions

None

Example

```
oracleForms.SetHeartbeat( 4 ); //sets the heartbeat intervals to 4 minutes.
```

SetProxy

This method is used only if the script is communicating with the Oracle Forms Server by HTTP. It enables the HTTP traffic to be routed through a proxy server.

Syntax

```
SetProxy( host, port)
```

Parameters

Parameter	Description
Host String	Host name or IP address of the proxy server.
Port int	Port number the proxy server is listening on.

Exceptions

```
QALoad Exception("Cannot set HTTP proxy after connection has been established.")
```

Example

```
oracleForms.SetProxy( "OurProxyServer", 80);
```

TabControl

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
TabControl( handlerID );
TabControl( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.TabControl( CONTROL_001 );
oracleFormsMsg1 = oracleForms.TabControl( "CONTROLabc" );
```

TextArea

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
TextArea( handlerID );
TextArea( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.TextArea( CONTROL_001 );
oracleFormsMsg1 = oracleForms.TextArea( "CONTROLabc" );
```

TextField

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
TextField( handlerID );
TextField( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.TextField( CONTROL_001 );
oracleFormsMsg1 = oracleForms.TextField( "CONTROLabc" );
```

Tlist

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
Tlist( handlerID );
Tlist( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.Tlist( CONTROL_001 );
oracleFormsMsg1 = oracleForms.Tlist( "CONTROLabc" );
```

Tree

This method constructs a new OracleFormsMsg for this type of control.

Syntax

```
Tree( handlerID );
Tree( controlName );
```

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The QALoad Script Development Workbench automatically creates constants for this value that refer to the specific control by name/type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>");
```

Example

```
oracleFormsMsg1 = oracleForms.Tree( CONTROL_001 );
oracleFormsMsg1 = oracleForms.Tree( "CONTROLabc" );
```

XmitMsg

Sends a message to the Oracle Forms Server.

Syntax

```
XmitMsg( statement, message )
```

Parameters

Parameter	Description
Statement int	A number used for script debugging and error reporting.
Message OracleFormsMsg	The message object to be sent.

Exceptions

```
QALoad Exception( "IOException called when transmitting message.")
```

Example

```
oracleForms.XmitMsg( 2, oracleFormsMsg1 );
```

XmitTerminalMessage

Sends a terminal message to the Oracle Forms Server.

A terminal message informs the server that the client has finished with data entry and that the server should process the previous set of data messages.

XmitTerminalMessage also waits for server responses. Automatic checkpointing takes place inside this method.

Syntax

```
XmitMsg( statement, responseCode )
```

Parameters

Parameter	Description
Statement int	A number used for script debugging and error reporting.
ResponseCode int	A number indicating the type of action to take. A value of 1 indicates that the previous messages should be acted on. A value of 3 indicates that the client wishes to disconnect from the server.

Exceptions

```
QALoad Exception( "IOException called when transmitting a terminal message.");
QALoad Exception( "IOException called when reading terminal message.");
QALoad Exception( " ** Server error: " + <message text> )
QALoad Exception( "Unknown control handle. ")
QALoad Exception( "Error expanding control array." )
```

Example

```
oracleForms.XmitTerminalMessage( 45, 1 );
```

QALoad

QALoad Index

BEGIN_TRANSACTION

Defines the beginning of the script's transaction loop.

BeginCheckpoint

Marks the beginning of a checkpoint.

CLOSE_ALL_DATA_POOLS

Closes all open local datapool files.

CLOSE_DATA_POOL

Closes the specified local datapool file.

COUNTER_VALUE

This command is used to update or increment the values of custom counters defined using the DEFINE_COUNTER command. As counter values are written to the timing file, they are time stamped with the elapsed time.

DATE_TIME

Gets the current time and/or date from the local machine.

DefaultCheckpointsOn

QALoad automatically adds this command when the Include Default Checkpoint statements convert option is selected in Workbench. When this command is found in a QALoad script, QALoad will not automatically generate checkpoints inside the middleware when Auto Timings is enabled in the QALoad Conductor. Instead, QALoad uses checkpoint statements found within the QALoad script.

DEFINE_COUNTER

Use the DEFINE_COUNTER command to define custom counters. Custom counters are written and managed on a per user basis. They will be saved to the timing file and can be graphed in Analyze. Counter data types can be either signed longs or floats. The counter type can be either cumulative or instance (which tells Analyze how to graph the counter.) Works in conjunction with the COUNTER_VALUE command.

DEFINE_TRANS_TYPE

Associates a description for the transaction loop displayed in QALoad Analyze.

DO_AbortOnError

Enables or disables error handling in the script.

DO_ExtractString

Finds a sub-string in a null-terminated buffer.

DO_SetTransactionCleanup

Defines a point at the end of the transaction for anything that needs to be deallocated or uninitialized. When transaction restarting occurs for a failed transaction, QALoad will first execute any code starting after the call to DO_SetTransactionCleanup allowing you to clean up important information and prevent memory leaks before retrying the transaction.

DO_SetTransactionStart

Defines a point at the beginning of the transaction loop that QALoad uses to rewind the transaction if the transaction fails and Restart Transaction error handling is selected in the QALoad Conductor.

DO_SetValue

Associates a value to a variable name. Variable names are embedded into parameter strings of QALoad functions and the value is interpolated at replay. Currently, DO_Http and DO_Https are the only functions that interpolate the variables.

DO_SLEEP

Inserts a sleep for the number of seconds defined in the parameter.

END_TRANSACTION

This command marks the end of the transaction loop.

EndCheckpoint

Indicates the end of a checkpoint, corresponding to a BeginCheckpoint command.

EXIT

Stops script processing and returns control back to the Conductor.

GET_ABSOLUTE_VUNUM

Gets the absolute virtual user number.

GET_DATA

Requests that QALoad Conductor send the next datapool record to the script.

GET_DATA_FIELD

Accesses the fields from the data record that was just read using the READ_DATA_RECORD statement. Field numbering starts at 1.

GET_DATAPOOLES_DIR

Retrieves the name of the QALoad Datapools directory.

GET_HOME_DIR

Retrieves the name of the QALoad installation directory.

GET_LOGFILES_DIR

Retrieves the name of the QALoad LogFiles directory.

GET_RELATIVE_VUNUM

Gets the relative virtual user number.

GET_SCRIPTS_DIR

Retrieves the name of the QALoad Scripts directory.

GET_TIMINGFILES_DIR

Retrieves the name of the QALoad Timing Files directory.

LOG_ERROR

Sends the corresponding message to the Conductor, so that it can be displayed within the [Player Messages](#) window in the Conductor.

OctalToChar

Converts any octal escape sequences to binary. Octal sequences consist of a backslash followed by two digits. This can be useful for adding binary data to a datapool file in the form of octal escape sequences since datapool files must contain only ASCII strings. For example:

OPEN_DATA_POOL

Opens the datapool file.

RANDOM_NUMBER

Returns a string representation of a pseudo-random random number.

RANDOM_STRING

Returns a string with a pseudo-random random set of alpha or alphanumeric characters of the specified width.

READ_DATA_RECORD

Reads a data record from a local datapool file.

RND_DELAY

Delays the script for a random interval before proceeding.

RND_DELAY_RANGE

Delays the script for a random interval, within a specified range, before proceeding.

RR_FailedMsg

Outputs a fatal error message to the Conductor.

RR_GetDebugFlag

Gets the debug flag for the script.

RR_printf

Prints formatted output to the standard output stream.

SET_ABORT_FUNCTION

Registers a callback function within the virtual user to call whenever the test operator manually aborts a

test from the QALoad Conductor.

SLEEP

Pauses a script for the specified number of seconds. This command is not affected by the sleep factor percentage specified in QALoad Conductor.

SYNCHRONIZE

Pauses script execution on the virtual user until the Conductor tells it to continue.

VARDATA

Replaces a string with a datapool variable.

BEGIN_TRANSACTION

Defines the beginning of the script's transaction loop.

QALoad automatically inserts BEGIN_TRANSACTION and END_TRANSACTION statements inside the script during the convert process. QALoad repeatedly executes the code between the BEGIN_TRANSACTION and END_TRANSACTION statements until you reach a maximum number of transactions or until the session duration time (specified in QALoad Conductor) is reached.

For each script, specify a frequency of execution with the pacing parameter in the QALoad Conductor. QALoad pauses the script after each transaction is complete, thereby ensuring that it does not send transactions to the system under test more rapidly than the pacing value specifies. This pause occurs at the BEGIN_TRANSACTION command.

Syntax

```
BEGIN_TRANSACTION( );
```

Parameters

None.

Example

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

BeginCheckpoint

Marks the beginning of a checkpoint.

You can turn enhanced checkpoints on or off from the QALoad Script Development Workbench's [Convert Options dialog box](#). BeginCheckpoint is always used in conjunction with an [EndCheckpoint](#) command.

Syntax

```
BeginCheckpoint (text);
```

Parameters

Parameter	Description
text	String containing a description of the checkpoint. This value cannot be

longer than 128 characters.

Example

```
BeginCheckpoint("Testing User-defined");
DO_Http("GET http://compuweb.compuware.com/ HTTP/ 1.0\r\n\r\n");
EndCheckpoint("Testing User-defined");
```

CLOSE_ALL_DATA_POOLS

Closes all open local datapool files.

All local datapool files should be closed at the end of the script using this statement.

Syntax

```
CLOSE_ALL_DATA_POOLS ();
```

Parameters

None.

Example

```
BeginCheckpoint();
RR_printf("Datapool Entry #1: %s", GET_DATA_FIELD 1);
DO_SLEEP(500);
EndCheckpoint(1);
CLOSE_ALL_DATA_POOLS ();
END_TRANSACTION( );
```

CLOSE_DATA_POOL

Closes the specified local datapool file.

All local datapool files should be closed at the end of the script using these statements or the `CLOSE_ALL_DATA_POOLS` command.

Syntax

```
CLOSE_DATA_POOL (int datapool ID);
```

Parameters

Parameter	Description
Datapool ID	The local datapool file to close.

Example

```
END_TRANSACTION();
CLOSE_DATA_POOL( SS_1 ); /* Default placement after */
/* END_TRANSACTION */
```

COUNTER_VALUE

Updates or increments the values of custom counters defined using the `DEFINE_COUNTER` command.

As counter values are written to the timing file, they are time stamped with the elapsed time.

Syntax

```
COUNTER_VALUE (int Counter_ID, long Counter_Value, float Counter_Value );
```

Parameters

Parameter	Description
Counter ID	A unique counter ID returned by a previous call to DEFINE_COUNTER.
Counter Value	The value for the counter.

Example

```
BEGIN_TRANSACTION();

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

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

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

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

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

DATE_TIME

Gets the current time and/or date from the local machine.

Syntax

```
char * DATE_TIME(const char *pformat);
```

Parameters

Parameter	Description
pformat	A formatted control string that determines how to format the date/time string. The following formats can be used: <ul style="list-style-type: none"> ! %a: Abbreviated weekday name ! %A: Full weekday name ! %b: Abbreviated month name ! %B: Full month name ! %c: Date and time representation appropriate for locale ! %d: Day of month as a decimal number (01-31) ! %H: Hour in a 24-hour format (00-23)

	! %I: Hour in a 12-hour format (01-12)
	! %j: Day of the year as a decimal number (001-366)
	! %m: Month as a decimal number (01-12)
	! %M: Minute as a decimal number (00-59)
	! %p: Current locale's AM/PM indicator for a 12-hour clock format
	! %S: Second as a decimal number (00-59)
	! %U: Week of the year as a decimal number, with Sunday as the first day of the week (00-53)
	! %w: Weekday as a decimal number (0-6, Sunday is 0)
	! %W: Week of the year as a decimal number, with Monday as the first day of the week (00-53)
	! %x: Date representation for the current locale
	! %X: Time representation for the current locale
	! %y: Year without a century, as a decimal number (00-99)
	! %Y: Year with a century, as a decimal number
	! %Z: Time zone name or abbreviation; no characters if the time zone is unknown
	! %: Percent sign

Example

```
char *temp = NULL;
temp = DATE_TIME("Today is %A, day %d of %B in the year %Y.");
free(temp);
//might produce the following:
//Today is Wednesday, day 21 of January in the year 2004.
```

DefaultCheckpointsOn

Automatically added when the Include Default Checkpoint statement's convert option is selected in Workbench.

When this command is found in a QALoad script, QALoad does not automatically generate checkpoints inside the middleware when Auto Timings is enabled in the QALoad Conductor. Instead, QALoad uses checkpoint statements found within the QALoad script.

Syntax

```
void DefaultCheckpointsOn (void);
```

Parameters

None


Example

```
...
// Checkpoints have been included by the convert process
DefaultCheckpointsOn ();
...
```

DEFINE_COUNTER

Defines custom counters.

Custom counters are written and managed on a per user basis. They are saved to the timing file and can be graphed in Analyze. Counter data types can be either signed longs or floats. The counter type can be either cumulative or instance, which tells Analyze how to graph the counter. Works in conjunction with the COUNTER_VALUE command.

 **Note:** If you call DEFINE_COUNTER more than once, with all of the same parameters, it returns the same counter ID.

Syntax

```
int DEFINE_COUNTER ( char* Group_Name, char* Counter_Name, char* Units, int Data_Type, int Counter_Type );
```

Parameters

Parameter	Description
Group Name	The name of the group this counter belongs to.
Counter Name	The name of the counter.
Units	The counter units. Can be NULL if no units are needed for this counter.
Data Type	Can be either DATA_LONG, or DATA_FLOAT.
Counter Type	Can be either COUNTER_CUMULATIVE, or COUNTER_INSTANCE.

Example

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

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

SYNCHRONIZE();
BEGIN_TRANSACTION();
```

The following is an example of a command to call each time an error occurs:

```
void ErrorOneOccurred()
{
int errorCounterID;
errorCounterID = DEFINE_COUNTER( "Some Error Group", "Error One", 0, DATA_LONG,
COUNTER_CUMULATIVE );
COUNTER_VALUE( errorCounterID, 1 );
}
```

DEFINE_TRANS_TYPE

Associates a description for the transaction loop displayed in QALoad Analyze .

Syntax

```
DEFINE_TRANS_TYPE ( "text" );
```

Parameters

Parameter	Description
text	A string of one to 60 characters enclosed in quotes.

Example

```
DEFINE_TRANS_TYPE ( "Receiving in Acquisition" );
```

DO_AbortOnError

Used to enable or disable error handling in a script.


The parameter that is passed to `DO_AbortOnError` sets how functions respond when an error is encountered. When an error is encountered, functions can continue or abort the script.

Under normal conditions, error handling is set in the Script Development Workbench (for validation), or in the Conductor. `DO_AbortOnError` overrides these product settings.

Syntax

```
DO_AbortOnError(flag);
```

Parameters

Parameter	Description
flag	<p>TRUE or FALSE. A flag indicating whether the script should abort upon receiving an error.</p> <p> Tip: If you choose FALSE, you must implement error checking for all functions that return a value to ensure that NULL or incorrect values are not subsequently used in the script.</p>

Example

```
char *p;
char temp[1000];
...
...
strcpy( temp, "Here is the search string." );

DO_AbortOnError(FALSE);
p = DO_GetUniqueStringEx( temp, "the", "string" );

DO_AbortOnError(TRUE);
if (p != NULL )
{
RR__printf( "String value = %s", p );
free( p );
}
else
{
//error handling if a NULL value is returned.
RR__printf( "String not found" );
}
}
```

DO_ExtractString

Finds a sub-string in a null-terminated data buffer.

Retrieves a unique value in the data buffer `szBuffer`. The parameters `szLeft` and `szRight` represent data just to the left and just to the right of a string in `szBuffer`. The `nCount` parameter specifies which left string to use if there is more than one matching string. The `pszResult` parameter is the address of a string (`char*`) that holds the resulting extracted string.

Syntax

```
BOOL DO_ExtractString(const char* szBuffer, int nCount, const char* szLeft, const char* szRight, char** pszResult);
```

Parameters

Parameter	Description
<code>szBuffer</code>	The buffer to search.
<code>nCount</code>	The number of occurrences in the left string before a match is made.
<code>szLeft</code>	The left side of the string to match.
<code>szRight</code>	The right side of the string to match.
<code>pszResult</code>	The address of the return string.

Example

```
char* szResult = 0;
...
DO_Http("GET http://www.host.com/ HTTP/1.1\r\n\r\n");
/*
 * The page returns a page containing "<title>Enter Login</title>"
 */
DO_ExtractString(DO_GetReplyBuffer(), 1, "<title>", "</title>", &szResult);
RR_printf("The extracted title: %s", szResult);
/*
 * prints "The extracted title: Enter Login"
 */
```

DO_SetTransactionCleanup

Defines a point at the end of the transaction for anything that needs to be deallocated or uninitialized.

When transaction restarting occurs for a failed transaction, QALoad first executes any code starting after the call to `DO_SetTransactionCleanup`, allowing you to clean up important information and prevent memory leaks before retrying the transaction. This function is used in conjunction with [DO_SetTransactionStart](#).

Syntax

```
DO_SetTransactionCleanup();
```

Parameters

None.

Example

```
BEGIN_TRANSACTION();
DO_SetTransactionStart();
TRANSACTION CODE...
DO_SetTransactionCleanup();
DO_HttpCleanup();
DO_SomeOtherMiddlewareCleanup();
END_TRANSACTION();
```

DO_MSLEEP

Pauses a script for the specified number of milliseconds.

The parameter passed to DO_MSLEEP is first scaled by the sleep factor percentage, which can be set from both the Workbench during validation and the Conductor at runtime.

During unit testing of the script, setting the sleep factor percentage to 0 (zero percent) causes DO_MSLEEP to not sleep at all. This command is ideal for unit testing where delays may not be wanted. Once the script is unit tested, the sleep factor percentage may be reset back to a suitable value, generally somewhere between 80% and 100%.

In addition, the sleep factor percentage can be set to Random in the QALoad Conductor. In this case, when a DO_MSLEEP command is encountered, it will sleep for a random time frame ranging from 0 to the value specified.

Syntax

```
DO_MSLEEP(time);
```

Parameters

Parameter	Description
time	Number of milliseconds to sleep before execution proceeds.

Example

This example shows how to pause a script for 50 milliseconds. This example sleeps 50 milliseconds if the sleep factor percentage is set to 100 in the QALoad Conductor .

```
DO_MSLEEP( 50 ); /* Sleep 50 milliseconds */
```

DO_SetTransactionStart

Defines a point at the beginning of the transaction loop that QALoad uses to rewind the transaction if the transaction fails and Restart Transaction error handling is selected in the QALoad Conductor. This function is used in conjunction with [DO_SetTransactionCleanup](#).

Syntax

```
DO_SetTransactionStart() ;
```

Parameters

None.

Example

```
BEGIN_TRANSACTION();
DO_SetTransactionStart();

TRANSACTION CODE...
```

QALoad 5.02

```
DO_SetTransactionCleanup();
DO_HttpCleanup();
DO_SomeOtherMiddlewareCleanup();
END_TRANSACTION();
```

DO_SetValue


Associates a value to a variable name.

Variable names are embedded into parameter strings of QALoad functions and the value is interpolated at replay. Currently, DO_Http and DO_Https are the only functions that interpolate the variables.

To embed a variable name, the name is wrapped by { and }. The default interpolation is to use the variable name as a part of the substituted value. For example, a name of "{this-name}" with a value of "this-value" is interpolated in the string "{this-name}" as "this-name=this-value". To suppress the variable name in the interpolated value, put an asterisk (*) right after the opening {. For example, a name of "this-name", with a value of "this-value" is interpolated in the string "{*this-name}" as "this-value".

After a variable is interpolated, it is removed from the variable table. For example, a name of "this-name" with a value of "this-value" is interpolated in the string "{*this-name} {*this-name}" as "this-value {this-name}".

If a variable is needed twice, it must be set twice. To suppress the removal of the variable from the variable table, put an exclamation (!) before the closing }. For example, a name of "this-name", with a value of "this-value" is interpolated in the string "{*this-name!} {*this-name!}" as "this-value this-value".

 **Note:** When using DO_SetValue to store CGI parameters, the parameters must be CGI encoded. This is done automatically by DO_GetFormValueByName, by the string constants inserted during conversion.

Syntax

```
BOOL DO_SetValue( const char *name, const char *value )
```

Parameters

Parameter	Description
name	String containing the name of the field in which to set a value.
value	String containing the value to set this field.

Example

```
...
...
DO_SetValue("name", "Joe+Smith" );
DO_SetValue("name", "Joe+Smith" );

DO_Http("GET http://company.com/forms.pl?{name} HTTP/1.0\r"
"\n Referer: http://company.com/forms.html\r\n Unused:"
"{*name}\r\n\r\n" );
```

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

QALoad will expand the statement internally as follows:

```
"GET http://company.com/forms.pl?name=Joe+Smith HTTP/1.0\r"
"\n Referer: http://company.com/forms.html\r\n"
"Unused: Joe+Smith\r\n\r\n"
```

DO_SLEEP

Inserts a sleep for the number of seconds defined in the parameter.

The parameter passed to DO_SLEEP is first scaled by the sleep factor percentage specified in QALoad Conductor. During unit testing of the script, setting the sleep factor percentage to 0 (zero percent) causes DO_SLEEP not to sleep at all.

This command is ideal for unit testing where delays may not be wanted. Once the script is unit tested, the sleep factor percentage may be reset back to a suitable value, generally somewhere between 80% and 100%.

In addition, the sleep factor percentage can be set to Random in the QALoad Conductor. In this case, when a DO_SLEEP command is encountered, it sleeps for a random time frame ranging from 0 to the value specified.

Syntax

```
DO_SLEEP( seconds );
```

Parameters

Parameter	Description
seconds	Number of seconds to sleep.

Example

This example shows how to pause a script for 5 seconds using the sleep function call. This example sleeps 5 seconds if the sleep factor percentage is set to 100 in the QALoad Conductor .

```
DO_SLEEP( 5 ); /* Sleep 5 seconds */
```

END_TRANSACTION

Marks the end of the transaction loop.

At the end of the transaction loop, the virtual user performs the following actions:

1. Records the transaction's elapsed time, from BEGIN_TRANSACTION to END_TRANSACTION. This is reported on the Analyze report as the Duration.
2. Determines if another transaction should be processed on this virtual user:
 - ! If the test is over, script processing continues with the command following the END_TRANSACTION.
 - ! If the test is not over, QALoad jumps to the BEGIN_TRANSACTION command, where the script is paused for pacing, if specified.

A test is over if one or more of the following conditions are met:

- ! The amount of time the test has been running exceeds the maximum session duration as set up in the session ID file.
- ! The operator has manually ended the test.
- ! This virtual user has executed the maximum number of transactions for the virtual users running this script as set on the Conductor's Script Assignment tab.

Syntax

```
END_TRANSACTION ( );
```

Parameters

None.

Example

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

EndCheckpoint

Indicates the end of a checkpoint, corresponding to a BeginCheckpoint command.

BeginCheckpoint and EndCheckpoint correspond to QALoad's enhanced checkpoints. You can turn enhanced checkpoints on or off from the QALoad Script Development Workbench's [Convert Options dialog box](#). EndCheckpoint is always used in conjunction with a [BeginCheckpoint](#) command.

Syntax

```
EndCheckpoint (text) ;
```

Parameters

Parameter	Description
text	String containing a description of the checkpoint. This value cannot be longer than 128 characters.

Example

```
BeginCheckpoint("Testing User-defined");  
DO_Http("GET http://compuweb.compuware.com/ HTTP/ 1.0\r\n\r\n");  
EndCheckpoint("Testing User-defined");
```

EXIT

Stops script processing and returns control back to the Conductor.

Syntax

```
EXIT ( );
```

Parameters

None.

Example

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

GET_ABSOLUTE_VUNUM

Gets the absolute virtual user number. This value is used to identify a virtual user uniquely within an entire test.

Syntax

```
int GET_ABSOLUTE_VUNUM ( );
```

Parameters

None.

Example

```
int vunum;
nuvum = GET_ABSOLUTE_VUNUM();
RR_printf("I am vu %d", vunum);
```

GET_DATA

Requests that QALoad Conductor send the next datapool record to the script.

If you reach the end of the datapool file when this command is called, the script either exits with an END OF DATA status in QALoad Conductor, or rewinds to the beginning of the datapool file, depending on the status of the rewind option in QALoad Conductor.

Syntax

```
GET_DATA ();
```

Parameters

None.

Example

```
BEGIN_TRANSACTION( ); /*Beginning of transaction loop*/
GET_DATA ();
...
RR_printf(VARDATA(1) );
```

GET_DATA_FIELD

Accesses the fields from the data record that were just read using the READ_DATA_RECORD statement. Field numbering starts at 1.

Syntax

```
GET_DATA_FIELD (datapool ID, FieldNum);
```

Parameters

Parameter	Description
Datapool ID	The datapool whose record should be used. This is necessary because you can have up to 32 local datapool files open at once.
FieldNum	Which field of the record to read. Field numbering starts at 1.

Example

```
BeginCheckpoint();
RR_printf("Datapool Entry #1: %s", GET_DATA_FIELD (1, 1) );
DO_SLEEP(500);
EndCheckpoint(1);
```

GET_DATAPOOLES_DIR

Retrieves the name of the QALoad Datapools directory.

For example, this function call returns the directory `\Program Files\Compuware\QALoad\Datapools`.

Syntax

```
const char *GET_DATAPOOLS_DIR()
```

Parameters

None.

Example

```
const char *pDatapoolsDir;
pDatapoolsDir = GET_DATAPOOLS_DIR();

// As an example, the default install directory for pDatapoolsDir would =
c:\ProgramFiles\Compuware\QALoad\ Datapools ;

// To print out the datapools directory, type
RR_printf("datapools directory = %s\n", GET_DATAPOOLS_DIR() ) ;
```

GET_HOME_DIR

Retrieves the name of the QALoad installation directory.

For example, this function call returns the directory \Program Files\Compuware\ QALoad .

Syntax

```
const char *GET_HOME_DIR()
```

Parameters

None.

Example

```
const char *pHomeDir;
pHomeDir = GET_HOME_DIR();

// As an example, the default installation directory for
// pHomeDir would = c:\Program Files\Compuware\ QALoad ;
```

GET_LOGFILES_DIR

Retrieves the name of the QALoad LogFiles directory.

For example, this function call will return the directory \Program Files\Compuware\QALoad\LogFiles.

Syntax

```
const char *GET_LOGFILES_DIR()
```

Parameters

None.

Example

```
const char *pLogFilesDir;
pLogFilesDir = GET_LOGFILES_DIR();

// As an example, the default installation directory for
// pLogFilesDir would = c:\Program Files\Compuware\QALoad\LogFiles;
```


GET_RELATIVE_VUNUM

Gets the relative virtual user number. This value is used to identify a virtual user uniquely within a player instance.

Syntax

```
int GET_RELATIVE_VUNUM ();
```

Parameters

None.

Example

```
int vunum;
nuvum = GET_RELATIVE_VUNUM();
RR_printf("I am vu %d", vunum);
```

GET_SCRIPTS_DIR

Retrieves the name of the QALoad Scripts directory.

For example, this function call will return the directory \Program Files\Compuware\QALoad\scripts.

Syntax

```
const char *GET_SCRIPTS_DIR()
```

Parameters

None.

Example

```
const char *pScriptsDir;
pScriptsDir = GET_SCRIPTS_DIR();

// As an example, the default installation directory for
// pScriptsDir would = c:\Program Files\Compuware\QALoad\Scripts;
```

GET_TIMINGFILES_DIR

Retrieves the name of the QALoad Timing Files directory.

For example, this function call will return directory \Program Files\Compuware\QALoad\TimingFiles.

Syntax

```
const char *GET_TIMINGFILES_DIR()
```

Parameters

None.

Example

```
const char *pTimingFilesDir;
pTimingFilesDir = GET_TIMINGFILES_DIR();

// As an example, the default installation directory for
// pTiming FilesDir would = c:\Program Files\Compuware\QALoad\TimingFiles ;
```

LOG_ERROR

Sends the corresponding message to the Conductor, so that it can be displayed within the [Player Messages](#) window in the Conductor.

Syntax

```
LOG_ERROR( int nSendMsg, char* msg );
```

Parameters

Parameter	Description
nSendMsg	Specifies whether msg should be sent to the Conductor.
msg	String that corresponds to the message to send to the Conductor.

Example

```
int rrobot_script( PLAYER_INFO *s_info )
{
    SET_ABORT_FUNCTION(abort_function);
    DEFINE_TRANS_TYPE( "CP01" );
    SYNCHRONIZE();
    BEGIN_TRANSACTION();
    LOG_ERROR(TRUE, "Message text here");
    END_TRANSACTION();
    REPORT(SUCCESS);
    EXIT();
    return(0);
}
```

OctalToChar

Converts any octal escape sequences to binary.

Octal sequences consist of a backslash followed by two digits. This can be useful for adding binary data to a datapool file in the form of octal escape sequences, since datapool files must contain only ASCII strings. For example:

\77 is equivalent to an ASCII 63 which is a question mark character.

\12 is equivalent to an ASCII 10 which is a linefeed character.

Syntax

```
int OctalToChar (char* str);
```

Parameters

Parameter	Description
str	A null-terminated string.

Example

```
char str[80];
...
strcpy(input, GET_DATA_FIELD(1, 1)); //copy data from datapool field into string variable
OctalToChar(input);
DO_WSK_Send(S1, input);
```

OPEN_DATA_POOL

Opens the datapool file.

This command line is typically placed before the BEGIN_TRANSACTION() statement.

Syntax

```
OPEN_DATA_POOL (frame, pNumber, rewindflag);
```

Parameters

Parameter	Description
frame	The name of the datapool file, including a drive and path.
pNumber	The ID that was given to the datapool when it was inserted into the script. This is used anytime the script reads a record or field from the datapool.
rewindFlag	TRUE if the datapool file should be rewound to the beginning after it reaches the end. FALSE if it should not rewind.

Example

```
OPEN_DATA_POOL( "C:\\Program Files\\Compuware\\ QALoad \\Middlewares\\
                SQLServer\\Scripts\\junk.dat", SS_1, TRUE );

/* Default placement before BEGIN_TRANSACTION */
SYNCHRONIZE();
BEGIN_TRANSACTION();
```

RANDOM_NUMBER

Returns a string representation of a pseudo-random random number between Low and High using Leading and Decimals to format the number.

The seed value that is used to generate the pseudo-random random number is set within the Conductor.

Syntax

```
char* RANDOM_NUMBER(int Low, int High, int Leading, int Decimals);
```

Parameters

Parameter	Description
Low	Lowest number that is generated.
High	Highest number that is generated.
Leading	If greater than zero, this value specifies how many digits must be present to the left of the decimal point. Values are padded with zeroes to reach the specified value.
Decimals	If greater than zero, this value specifies how many digits must be present to the right of the decimal point. If zero is specified, no decimal point is generated.

Example

```
char *temp = NULL;
temp = RANDOM_NUMBER(1, 100, 3, 2);
free(temp);
//might produce the following strings:
// "004.38"
// "319.03"
// "077.12"
```

RANDOM_STRING

Returns a string with a pseudo-random random set of alpha or alphanumeric characters of the specified width.

The seed value that is used to generate the pseudo-random random number is set within the Conductor.

Syntax

```
char* RANDOM_STRING(int AlphaNum, int WidthMin, int WidthMax);
```

Parameters

Parameter	Description
AlphaNum	One of the following values: 0: Returning string should contain only numeric values 1: Returning string should contain only alpha values 2: Returning string should contain alpha and numeric values
WidthMin	Minimum width of the variable width format of the call.
WidthMax	Maximum width of the variable width format of the call.

Example

```
char *temp = NULL;
temp = RANDOM_STRING(1, 4, 10);
free(temp);
//might produce the following strings:
// "fj32"
// "mfigkec973"
// "fik34kf"
```

READ_DATA_RECORD

Reads a data record from a local datapool file.

This statement is typically placed after the BEGIN_TRANSACTION statement, although it is possible to read more than one record from the file during a single transaction.

Syntax

```
READ_DATA_RECORD(datapool ID);
```

Parameters

Parameter	Description
-----------	-------------

Datapool ID	Tells from which local datapool file to read the record.
-------------	--

Example

```
BEGIN_TRANSACTION();
READ_DATA_RECORD( SS_1 ); /* Default placement - Start of */
/* Transaction loop */
```

RND_DELAY

Delays the script for a random interval before proceeding.

Each time the script executes the RND_DELAY command, the Player generates a random number. It uses a uniform distribution, between 0 and n seconds where n is the parameter to the RND_DELAY command. The average delay time for multiple occurrences of this command is n/2 seconds.

Syntax

```
RND_DELAY ( n );
```

Parameters

Parameter	Description
n	Maximum number of seconds to delay before script execution proceeds.

RND_DELAY_RANGE

Delays the script for a random interval, within a specified range, before proceeding.

Each time the script executes the RND_DELAY_RANGE command, the Player generates a random number. It uses a uniform distribution between minTime and maxTime seconds.

Syntax

```
int RND_DELAY_RANGE (int minTime, int maxTime);
```

Parameters

Parameter	Description
minTime	Minimum number of seconds to delay before script execution continues.
maxTime	Maximum number of seconds to delay before script execution continues.

Example

In this example, the script pauses for a pseudo-random range between 2 and 10 seconds using the random delay range function.

```
RND_DELAY_RANGE(2, 10); /* Sleep between 2 and 10 seconds. */
```

RR_FailedMsg

Outputs a fatal error message to the Conductor. Use this function to describe an error condition encountered that caused the script to fail.

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

Syntax

```
int RR__FailedMsg (PLAYERINFO *, char *msg);
```

Parameters

Parameter	Description
PLAYERINFO	Pointer to the PLAYERINFO struct, <code>sinfo</code> .
msg	Message to be passed to the Conductor.

Example

```
int ret = 0;
ret = myFunc();
if(ret == ERROR)
RR__FailedMsg(s_info, "Virtual User Failed on myFunc!");
```

RR__GetDebugFlag

Gets the debug flag for the script.

Syntax

```
int RR__GetDebugFlag ();
```

Parameters

none

Example

```
RR__GetDebugFlag ();
```

RR__printf

Prints formatted output to the standard output stream.

`RR__printf` formats and prints a series of characters and values to the standard output stream, `stdout`. If arguments follow the format string, the format string must contain specifications that determine the output format for the arguments.

The format argument consists of ordinary characters, escape sequences, and, if arguments follow format, format specifications. The ordinary characters and escape sequences are copied to `stdout` in order of their appearance.

For example, the line:

```
RR__printf("Line one\n\t\tLine two\n");
```

produces the output:

```
Line one
Line two
```

Format specifications always begin with a percent sign (%) and are read left to right. When `RR__printf` encounters the first format specification, if any, it converts the value of the first argument after format and outputs it accordingly. The second format specification causes the second argument to be converted and

output, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for all the format specifications.

Syntax

```
int RR_printf(const char * format [, argument]...);
```

Parameters

Parameter	Description
format	Format control.
argument	Optional arguments.

Example

```
/* PRINTF.C: This program uses the printf and wprintf functions to produce formatted output.
 */
#include <stdio.h> void main(void)
{
char ch='h', *string="computer";
int count=-9234;
double fp=251.7366;
wchar_t wch=L'w', *wstring=L"Unicode";

/*Display integers. */
printf("Integer formats:\n" "\tDecimal: %d Justified: %.6d Unsigned: %u\n", count, count,
count, count);

printf("Decimal %d as:\n\tHex: %Xh C hex: 0x%x Octal: %o\n", count, count, count, count);

/* Display in different radixes. */
printf("Digits 10 equal:\n\tHex: %i Octal: %i Decimal: %i\n",0x10, 010, 10);

/* Display characters. */
printf("Characters in field (1):\n%10c%5hc%5c%5lc\n", ch, ch, wch, wch);
wprintf(L"Characters in field (2):\n%10C%5hc%5c%5lc\n", ch, ch, wch, wch);

/* Display strings. */
printf("Strings in field (1):\n%25s\n%25.4hs\n\t%S%25.3ls\n", string, string, wstring,
wstring);

wprintf(L"Strings in field (2):\n%25S\n%25.4hs\n\t%S%25.3ls\n", string, string, wstring,
wstring);

/* Display real numbers. */
printf("Real numbers:\n\t%f%.2f%e%E\n", fp, fp, fp, fp);

/* Display pointer. */
printf("\nAddress as:\t%p\n", &count);

/* Count characters printed. */
printf("\nDisplay to here:\n");
printf("1234567890123456%n78901234567890\n", &count);
printf("\tNumber displayed: %d\n\n", count);
}
```

Output

Integer formats:

```
Decimal: -9234
Justified: -009234
Unsigned: 4294958062
```

Decimal -9234 as:

QALoad 5.02

Hex: FFFFD BEEh
C hex: 0xffffdbee
Octal: 37777755756

Digits 10 equal:

Hex: 16
Octal: 8
Decimal: 10

Characters in field (1):

h h w w

Characters in field (2):

h h w w

Strings in field (1):

computer
comp
Unicode Uni

Strings in field (2):

computer
comp
Unicode Uni

Real numbers:

251.736600
251.74 2.517366e+002
2.517366E+002

Address as:

0012FFAC

Display to here:

123456789012345678901234567890


Number displayed:

16

SET_ABORT_FUNCTION

Registers a callback function within the virtual user to call whenever the test operator manually aborts a test from the QALoad Conductor.

When the abort callback function returns, the script automatically exits.

 **Note:** Checkpoints executed during an abort are not recorded in the timing file.

Syntax

```
SET_ABORT_FUNCTION ( functionName );
```

Parameters

Parameter	Description
functionName	Name of a function to call when the test is aborted.

Example

```
{
/* Script Initialization */
:
SET_ABORT_FUNCTION( abort_function ) ;
/* Script */
}

void abort_function( PLAYER_INFO * s-info ) ;

{
/* Abort functionality goes here */
EXIT( ) ;
}
```

SCRIPT_MESSAGE

The **SCRIPT_MESSAGE** command inserts custom script messages into a timing file during test execution. The command takes a group name and a message as parameters; the messages appear on the Error report in Analyze when they are used.

Syntax

```
SCRIPT_MESSAGE ( char* group, char* msg );
```

Parameters

Parameter	Description
group	Message group name.
msg	Message.

Example

```
int rrobot_script( PLAYER_INFO *s_info )
{
    SET_ABORT_FUNCTION(abort_function);
    DEFINE_TRANS_TYPE( "CP01" );
    SYNCHRONIZE();

    BEGIN_TRANSACTION();
    SCRIPT_MESSAGE("My Group", "Message text here");
    END_TRANSACTION();
    REPORT(SUCCESS);
    EXIT();
    return(0);
}
```

SLEEP

Pauses a script for the specified number of seconds.

This command is not affected by the sleep factor percentage specified in QALoad Conductor.

Syntax

```
SLEEP ( n );
```

Parameters

Parameter	Description
n	The number of seconds to sleep before execution proceeds.

SYNCHRONIZE

Pauses script execution on the virtual user until the Conductor tells it to continue.

Normal usage is to have all scripts synchronize when they have reached the point at which transaction processing is to begin.

There can be only one SYNCHRONIZE command per script.

Syntax

```
SYNCHRONIZE( );
```

Parameters

None.

Example

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

SYNCH

This command is used to synchronize all virtual users for a particular script. When this statement is reached, QALoad halts execution of the virtual user. The virtual user remains halted until all other virtual users for this script have also halted at this statement. Then, the Conductor instructs all virtual users to continue.

Unlike the SYNCHRONIZE command, which is automatically added to the script above the BEGIN_TRANSACTION statement by the convert process, you can insert any number of SYNCH commands into a script.

Syntax

```
SYNCH( );
```

Parameters

None.

Example

```
...
...
SYNCHRONIZE( );
...
BEGIN_TRANSACTION( );
...
SYNCH( );
...
...
END_TRANSACTION( );
```

VARDATA

Replaces a string with a datapool variable.

To insert data from the fields in a datapool, substitute VARDATA(n) expressions wherever you want to replace a string with variable data. Note that datapool field numbering starts at 1.

Syntax

```
VARDATA(n)
```

Parameters

Parameter	Description
n	The datapool field number. Field numbering starts at 1.

Example

```
Do_TuxFMLData ( 8302, 0, VARDATA(1));
```

QARun integration

QARun Integration Index

DO_InitAWL

Initializes internal variables needed to support QARun integration.

DO_StartAWL

Starts the 32-bit QARun script indicated in the parameters with the specified username, password, and environment from the QARun database.

DO_InitAWL

Initializes internal variables needed to support QARun integration.

Insert this command at the beginning of a QALoad script that calls a QARun test script.

Syntax

```
DO_InitAWL(s_info);
```

Parameters

Parameter	Description
s_info	Structure used by each virtual user.

Example

```
DO_InitAWL(s_info);
```

DO_StartAWL

Starts the 32-bit QARun script indicated in the parameters with the specified username, password, and environment from the QARun database.

If the function is successful, it returns a zero (0). If the function is not successful, it returns a one (1).

Syntax

```
DO_StartAWL(AWLUsername, AWLEnv, AWLScriptName);
```

Parameters

Parameter	Description
AWLUsername	A string value containing the QARun database username and password, separated by a comma.
AWLEnv	A string value containing a valid QARun script environment from the QARun database.
AWLScriptName	A string value containing the name of an existing QARun script in the QARun database.

Example

```
nFail = DO_StartAWL ("admin,admin", "default", "script1");
```

SAP 4.x

SAP 4.x Index

Do_SAPCheckScreen

Used as a synchronization point in your QALoad script.

Do_SAPCheckStatus

Do_SAPCheckStatus compares the status message at the bottom of the SAP window with an expected message fragment.

Do_SAPCheckTitle

The Do_SAPCheckTitle function compares the SAP screen title with some expected text.

Do_SAPClearMessages

Do_SAPClearMessages checks the title of the SAP window and the SAP status message at the bottom.

Do_SAPDumpEvent

Prints the current values of the IT_EVENT structure to the Player window.

Do_SAPExtractString

Allows you to specify a string to search (src), a filter string, and a left and right filter marker, and will return the number of characters copied into the destination variable (dest).

Do_SAPFrontEnd

Enables graphical display for replay of scripts.

Do_SAPFullMenu

Enables full or dynamic menus during SAP logins.

Do_SAPGetControlValue

Used to retrieve the value for the specified control.

[Do_SAPGetIt_Event](#)

Retrieves a pointer to the IT_EVENT structure (used in GUILIB.H).

[Do_SAPGetScreenTitle](#)

Retrieves a pointer to the current screen title from the current IT_EVENT structure.

[Do_SAPGetStatusMsg](#)

Retrieves a pointer to the status message from the current IT_EVENT structure.

[Do_SAPInit](#)

This function is required, and initializes the thread local storage for each SAP thread.

[Do_SAPLogging](#)

Turns the detail logging capability on and off.

[Do_SAPLogin](#)

Connects and logs into the SAP application server. If the script is unable to connect to the server, an error message is displayed and the script is aborted.

[Do_SAPLogoff](#)

Logs the script off of the application server.

[Do_SAPSendDbfClick](#)

Whenever the user double-clicks when using the SAP GUI during capture, a Do_SAPSendDbfClick event is recorded to the capture file.

[Do_SAPSendEvent](#)

Causes the script to send the current event to server.

[Do_SAPSendMenu](#)

Locates the Menu name provided in the Menu string and sets the menu option in the current event structure. The menu string for multi-level menus is formed by appending a -> string to each menu level, such as Edit->Copy, or File->open. It then sends the event to the server.

[Do_SAPSendOKCode](#)

Sets the OK code field in the IT_EVENT structure. The event is then automatically sent to the server.

[Do_SAPSendPFKey](#)

Duplicates the sending of a PFKey by the application by setting the PFKey value in the IT_EVENT structure and then sending the transaction to the server.

[Do_SAPSendReturn](#)

Sets the VK_RETURN PFKey in the event structure and sends the event to the server. This is the same as pressing Return or as the Green Check mark on the toolbar.

[Do_SAPSetCheckScreenWildcard](#)

Specifies the wildcard character to use for wildcard matching with Do_SAPCheckScreen.

[Do_SAPSetCtrlValue](#)

Sets the value for a control.

[Do_SAPSetCursor](#)

Sets the cursor fields in the IT_EVENT structure to the specified control.

Do_SAPCheckScreen

Used as a synchronization point in your QALoad script. Should be called after each Do_SAPSendEvent() to insure that the screen returned by the server is the one expected by the script.

Each event sent from the SAP application server to the client includes the name of the ABAP program and the current screen title. This command ensures that the script and the application do not get out of sync.

Use Do_SAPCheckScreen with Do_SAPSetCheckScreenWildcard to perform a wildcard search for a screen title. This is especially useful if the screen title is likely to change with each new entry/lookup in the database during replay. If you insert the wildcard (set in Do_SAPSetCheckScreenWildcard) as the first character of the title, then all titles with the same right-most characters will match. If the wildcard is located in any character position other than the first, QALoad does not treat it as a wildcard. For example, "*est" will match "test," "tempest," and "est," but will not match "tester." This prevents possible conflicts when a wildcard character is present in a captured string, but is not intended to be a wildcard. This also prevents conflicts within pre-existing scripts that were converted before the wildcard matching option was added in Release 4.4.

If the end of a title is problematic during replay, it isn't necessary to use a wildcard match. Instead, reduce the number of characters that are compared in the title. For example, if the order number in the title "ORDER# PROCESSED: 12345" is likely to change during replay, shorten the title to remove the characters that are changing. For example, shorten the title to "ORDER# PROCESSED:". This will result in a match with any title during replay that contains the first characters "ORDER# PROCESSED:".

Syntax

```
Do_SAPCheckScreen ( OKCode, ScreenName, title );
```

Parameters

Parameter	Description
OKCode	A string containing the current transaction code.
ScreenName	A string containing the current screen name.
title	A string containing the current string title.

Example

```
Do_SAPCheckScreen( "MMR1", "SAPLMGMM", "Create Raw Material: Initial Screen" );
```

Do_SAPCheckStatus

Compares the status message at the bottom of the SAP window with an expected message fragment.

The second parameter of the function is the display flag; TRUE prints out the status message when it is unexpected.

Syntax

```
Do_SAPCheckStatus( szExpected );
```

Parameters

Parameter	Description
szExpected Char *	A string containing the current string status.

Example

```
Do_SAPSendPFKey("Save"); // Click the Save button
CHECK_POINT( TIMER9, 33); // Record response time for saving document
if (!Do_SAPCheckStatus("has been saved")) abort_function();
// Abort due to an
//unexpected status
//message
```

Do_SAPCheckTitle

Compares the SAP screen title with some expected text.

Syntax

```
Do_SAPCheckTitle( szExpected );
```

Parameters

Parameter	Description
szExpected Char *	A string containing the current string title.

Example

```
Do_SAPSendReturn();
if (Do_SAPCheckTitle("PO Extension")) Do_SAPSendReturn();
// This screen may sometimes appear. Press Return if it does.
```

Do_SAPClearMessages

Checks the title of the SAP window and the SAP status message at the bottom.

If the window title is "Information", then an information popup box has appeared; a return is sent to clear it. If the status message includes "W:", then a warning has appeared. Press Return to clear the warning. Do_SAPClearMessages loops until no more information boxes or warnings appear.

Syntax

```
Do_SAPClearMessages( );
```

Parameters

None.

Example

```
Do_SAPSetCtrlValue ( "VBAK-AUART", 0, "yta " );
// Enter data into SAP field.

Do_SAPSendReturn();
// Press Return to validate input.

Do_SAPClearMessages();
// Clear any warnings and information boxes
// and display the messages.

Do_SAPSendPFKey("Save");
// Click the Save button.

CHECK_POINT( TIMER9, 1);
// Record response time for saving document.
```

Do_SAPDumpEvent

Prints the current values of the IT_EVENT structure to the Player window.

This command is used for debugging purposes, primarily to view control and function key names and values.

Syntax

```
Do_SAPDumpEvent();
```

Parameters

None.

Example

```
Do_SAPDumpEvent();
```

Do_SAPExtractString

Allows you to specify a string to search (src), a filter string, and a left and right filter marker, and returns the number of characters copied into the destination variable (dest).

Syntax

```
int Do_SAPExtractString ( dest, src, filter, leftmarker, rightmarker );
```

Parameters

Parameter	Description
dest Char *	Pointer to destination string.
src Char *	Pointer to string to search.
filter Char *	Pointer to filter string.
leftmarker Unsigned short	Left filter marker.
rightmarker Unsigned short	Right filter marker.

Example

```
char src[64];
char dest[64];
char filter[64];
strcpy ( src, "Inventory number = 123456" );
strcpy ( filter, "Inventory number = <>" );
Do_SAPExtractString ( dest, src, filter, '<', '>' );
printf ( "dest = [%s]\n", dest );
```

Output

```
"dest = [123456]"
```

Do_SAPFrontEnd

Enables graphical display for replay of scripts.

Syntax

```
Do_SAPFrontEnd( long flag );
```

Parameters

Parameter	Description
flag	Full menu flag. Valid values are: TRUE: Enables SAP graphical display FALSE: Disables SAP graphical display

Example

```
Do_SAPFrontEnd( TRUE );
```

Do_SAPFullMenu

Enables full or dynamic menus during SAP logins.

Syntax

```
Do_SAPFullMenu( flag );
```

Parameters

Parameter	Description
flag	TRUE causes SAP logins with full menus (all the menu structure is downloaded from the application server at once). FALSE causes SAP logins with dynamic menus (menus are downloaded as the menus are used.)

Example

```
Do_SAPFullMenu( FALSE );
```

Do_SAPGetControlValue

Used to retrieve the value for the specified control.

The control is specified by name, with the occurrence parameter distinguishing between multiple controls of the same name. The Do_SAPDumpEvent() command can be used to display a list of controls on the current form. If the control is a radio button or a check button, the literals TRUE or FALSE are placed in the buffer indicating if the control is currently selected.

Syntax

```
int Do_SAPGetControlValue( CtrlName, CtrlOcc, Buffer );
```

Parameters

Parameter	Description
CtrlName string	Contains the name of the control.
CtrlOcc integer	Shows the occurrence of the control (0=first occurrence).

Buffer Char *	A pointer to a buffer to receive the current contents of the control. Note that no length checking is performed, so ensure that the buffer is large enough to contain the returned value.
---------------	---

Example

```
char buff[1024];
Do_SAPGetControlValue ( "RMMG1-MATNR", 1, buff );
```

Do_SAPGetIt_Event

Retrieves a pointer to the IT_EVENT structure (used in GUILIB.H).

Syntax

```
PIT_EVENT Do_SAPGetIt_Event();
```

Parameters

None.

Example

```
PIT_EVENT temp;
temp = Do_SAPGetIt_Event ();
printf ("\nScreen Title = %s\n", temp->szNormTitle);
```

Do_SAPGetScreenTitle

Retrieves a pointer to the current screen title from the current IT_EVENT structure.

Syntax

```
char *Do_SAPGetScreenTitle ( );
```

Parameters

None.

Example

```
char *Msg;
Msg = Do_SAPGetScreenTitle ( );
```

Do_SAPGetStatusMsg

Retrieves a pointer to the status message from the current IT_EVENT structure.

This command is typically used after a Do_SAPSendEvent() statement to check for possible returned error messages.

Syntax

```
char *Do_SAPGetStatusMsg();
```

Parameters

None.

Example

```
char *Msg;
Msg = Do_SAPGetStatusMsg();
```

Do_SAPInit

This function is required, and initializes the thread local storage for each SAP thread.

Syntax

```
Do_SAPInit( s_info );
```

Parameters

Parameter	Description
s_info	Structure used by each virtual user.

Example

```
Do_SAPInit( s_info );
```

Do_SAPLogging

Turns the detail logging capability on and off.

Logging is a debugging tool that logs all SAP events received from the server to a log file named saplg###.log, where ### is the virtual user number on the player. The log file is a binary file that can only be viewed using the QALSAP program. The flag parameter is used to turn logging off and on or to clear simultaneously any previous log entries. Once logging is turned on, a log record is written before each event is sent to the SAP server and after each event is received from the server.

Syntax

```
Do_SAPLogging( flag );
```

Parameters

Parameter	Description
flag integer	One of the following constants defined in Do_SAP.H: SAPLOG_ON SAPLOG_OFF SAPLOG_CLEAR

Example

```
Do_SAPLogging( SAPLOG_ON );
```

Do_SAPLogin

Connects and logs into the SAP application server. If the script is unable to connect to the server, an error message is displayed and the script is aborted.

Syntax

```
Do_SAPLogin( hostname, systemNo, clientNo, Name, Pwd, Lang);
```

Parameters

Parameter	Description
hostname string	The Host name of the SAP application server.
systemNo string	The system number to login to.

clientNo string	The client number to use (may be defaulted to "").
Name string	The user ID.
Pwd string	The password.
Lang string	The language (may be defaulted to "").

Example

```
Do_SAPLogin( "/H/204.79.199.5/H/207.213.200.19", "01", "850", "compuware", "qaload", "" );
```

Do_SAPLogoff

Logs the script off of the application server.

Syntax

```
Do_SAPLogoff();
```

Parameters

None.

Example

```
Do_SAPLogoff();
```

Do_SAPSendDbfClick

Whenever the user double-clicks when using the SAP GUI during capture, a Do_SAPSendDbfClick event is recorded to the capture file.

Syntax

```
Do_SAPSendDbfClick ( );
```

Parameters

None.

Example

```
Do_SAPSetCursor( "RMMG1-MBRSH", 0 ); /* Value: Industry s */
Do_SAPSendDbfClick();
```

Do_SAPSendEvent

Causes the script to send the current event to the server.

Typically, the script sets the value of one or more controls and indicates the action to be taken by setting a function key using Do_SAPPrKey(). The Do_SAPSendEvent function completes the screen by transmitting the updated information and action code to the server. Upon completion of the Do_SAPSendEvent() function, the script automatically retrieves the server's reply.

Syntax

```
Do_SAPSendEvent();
```

Parameters

None.

Example

```
Do_SAPSendEvent();
```

Do_SAPSendMenu

Locates the Menu name provided in the Menu string and sets the menu option in the current event structure.

The menu string for multi-level menus is formed by appending a "->" string to each menu level, such as Edit->Copy, or File->open. It then sends the event to the server.

Syntax

```
Do_SAPSendMenu( MenuString );
```

Parameters

Parameter	Description
MenuString string	The (case insensitive) name of the menu option to set.

Example

```
Do_SAPSendMenu( "Log off" );
```

Do_SAPSendOKCode

Sets the OK code field in the IT_EVENT structure. The event is then automatically sent to the server.

Syntax

```
Do_SAPSendOKCode ( OKcode );
```

Parameters

Parameter	Description
OKCode string	A string containing the new OK code to send.

Example

```
Do_SAPSendOKCode( "/nmmr1" );
```

Do_SAPSendPFKey

Duplicates the application sending a PFKey by setting the PFKey value in the IT_EVENT structure and then sending the transaction to the server.

The key name supplied as a parameter to Do_SAPSendPFKey() must be present in the current screen's IT_EVENT structure. If it isn't, an error message is given and the script is aborted.

To send a value that isn't one of the currently available key names, call this function with the decimal virtual key value (iVKValue) preceded by a #. For example, to send an F1 whether or not it was defined in the IT_EVENT structure, call Do_SAPSendPFKEY("#112").

Syntax

```
Do_SAPSendPfKey ( KeyName );
```

Parameters

Parameter	Description
KeyName string	The name of the key to set (case insensitive).

Example

```
Do_SAPSendPFKey ( "Possible entries" );
```

Do_SAPSendReturn

Sets the VK_RETURN PFKey in the event structure and sends the event to the server. This is the same as pressing Return or as the Green Check mark on the toolbar.

Syntax

```
Do_SAPSendReturn();
```

Parameters

None.

Example

```
Do_SAPSendReturn();
```

Do_SAPSetCheckScreenWildcard

Specifies the wildcard character to use for wildcard matching with Do_SAPCheckScreen.

Although all converted scripts include Do_SAPSetCheckScreenWildcard ('*') as one of the first functions, you can specify a different wildcard character to use later in the script.

Syntax

```
Do_SAPSetCheckScreenWildcard ( wildcard );
```

Parameters

Parameter	Description
wildcard unsigned short	A character to match in Do_SAPCheckScreen.

Example

```
Do_SAPSetCheckScreenWildcard ( '*' );
```

Do_SAPSetCtrlValue

Sets the value for a control.

If the control is a radio-button, then all other radio-buttons in that control's group are cleared.

Syntax

```
Do_SAPSetCtrlValue ( CtrlName, CtrlOcc, Buffer );
```

Parameters

Parameter	Description
CtrlName string	The name of the control.
CtrlOcc integer	Occurrence of the control (0=first occurrence).
Buffer Char *	Pointer to a buffer containing the value to load into the control. If the control is a radio button or a check button, then the buffer must contain the string TRUE to set the button, or FALSE to clear the button.

Example

```
Do_SAPSetCtrlValue ( "RMMG1-MATNR", 1, "B123458" );
```

Do_SAPSetCursor

Sets the cursor fields in the IT_EVENT structure to the specified control.

If the control appears multiple times on the screen, then the ctrlOcc field can be used to specify which control the cursor is to be set to. CtrlOcc is zero-based, that is, the first occurrence is 0, then the second is 1, and so forth. To set the cursor to an arbitrary point on the screen, the ctrlName parameter may be entered in a numeric format "#row,col (i.e. "#2,1").

Syntax

```
Do_SAPSetCursor( ctrlName, ctrlOcc );
```

Parameters

Parameter	Description
ctrlName string	The name of the control to set the cursor to.
ctrlOcc integer	The occurrence of the control name.

Example

```
Do_SAPSetCursor( "MSICHTAUSW-KZSEL ", 0 ); /* Value: Variant */
```

SAP 6.x

SAP 6.x Index

SAPGui_error_handler

Outputs a fatal error message to the Conductor, which causes the virtual user to either fail or report a warning.

SAPGuiApplication

Allows scripts to call SAP GUI low-level administrative objects of the SAP GUI.

SAPGuiCheckScreen

Acts as a synchronization point in the script.

SAPGuiCheckStatusbar

Specifies the method (or property) that is called (or set), allowing the script access to the SAP GuiStatusbar object.

[SAPGuiCmd0](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. No parameters are sent.

[SAPGuiCmd1](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. One parameter is sent.

[SAPGuiCmd1Coll](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. One parameter is sent. Use this variation to deal with Collection object information only.

[SAPGuiCmd1Elmnt](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. One parameter is sent. This variation is to be used for entering COM array element info only (VB Collections).

[SAPGuiCmd1Sub](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. One parameter is sent. This variation is to be used for dealing with subtype information only.

[SAPGuiCmd1Sub1](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. One parameter is sent. This variation is to be used for dealing with subtype and SubParameter information only.

[SAPGuiCmd2](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. Two parameters are sent.

[SAPGuiCmd3](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. Three parameters are sent.

[SAPGuiConnect](#)

Specifies to which server a connection should be made.

[SAPGuiCreateColl](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. This call creates a collection.

[SAPGuiDestroyColl](#)

Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. This call destroys a collection and decrements the reference count.

[SAPGuiPropIdStr](#)

Specifies the object ID string to use with subsequent SAPGui calls.

[SAPGuiPropIdStrExists](#)

Specifies the object ID string to use with subsequent SAPGui calls.

[SAPGuiPropIdStrExistsEnd](#)

Marks the end of the block of code that is executed if the condition in a prior SAPGuiPropIdStrExists command is true.

[SAPGuiSessionInfo](#)

Specifies the method (or property) that will be called, allowing the script access to the SAP GuiSessionInfo objects.

[SAPGuiSetCheckScreenWildcard](#)

Specifies the wildcard character to use for wildcard matching with SAPGuiCheckScreen.

SAPGuiVerCheckStr

Specifies the SAP GUI frontend version number at the time the capture file was made.

SAPGui_error_handler

Outputs a fatal error message to the Conductor, which causes the virtual user to either fail or report a warning.

Syntax

```
SAPGui_error_handler(PLAYERINFO *, char *msg);
```

Parameters

Parameter	Description
PLAYERINFO	Pointer to the PLAYERINFO struct, sinfo
msg	Message to pass to the Conductor.

Example

```
{
    char buffer[1024];
    sprintf(buffer, "SAP: error occurred for VU(%i)\n", S_task_id);
    SAPGui_error_handler(s_info, buffer);
}
```

SAPGuiApplication

Allows scripts to call low-level administrative objects of the SAP GUI.

Syntax

```
SAPGuiApplication(FuncName);
```

Parameters

Parameter	Description
FuncName	Function name.

Example

```
.
.
.

BEGIN_TRANSACTION();
DO_SetTransactionStart();
try{
    SAPGuiConnect( s_info, "qacsapdb");
    SAPGuiApplication(RegisterROT);
    SAPGuiVerCheckStr("6402.160.1");
.
.
.
```

SAPGuiCheckScreen

Used as a synchronization point in your QALoad script.

Call this command after each request block to ensure that the screen being returned by the server is the one expected by the script. Each event sent from the SAP application server to the client includes the name of the ABAP program and the current screen title. This command ensures that the script and the application remain in synch.

Use `SAPGuiCheckScreen` with `SAPGuiSetCheckScreenWildcard` to perform a wildcard search for a screen title. This is especially useful if the screen title is likely to change with each new entry/lookup in the database during replay.

If you insert the wildcard (set in `SAPGuiSetCheckScreenWildcard`) as the first character of the title, then all titles with the same right-most characters will match. If the wildcard is located in any character position other than the first, QALoad does not treat it as a wildcard. For example, `*est` will match `test`, `tempest`, and `est`, but will not match `tester`. This prevents possible conflicts when a wildcard character is present in a captured string, but is not intended to be a wildcard. This also prevents conflicts within pre-existing scripts that were converted before the wildcard matching option was added in Release 4.4.

If the end of a title is problematic during replay, it is not necessary to use a wildcard match. Instead, reduce the number of characters that are compared in the title. For example, if the order number in the title `ORDER# PROCESSED: 12345` is likely to change during replay, shorten the title to remove the characters that are changing. In this case, shorten the title to `ORDER# PROCESSED:`. This results in a match with any title during replay that contains the first characters `ORDER# PROCESSED:`.

Syntax

```
SAPGuiCheckScreen ( OKCode, ScreenName, title );
```

Parameters

Parameter	Description
OKCode	A string containing the current transaction code.
ScreenName	A string containing the current screen name.
title	A string containing the current string title.

Example

```
SAPGuiCheckScreen( "SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access" );

//Check the OKcode, ScreenName, and screen title after each command

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

SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,94,24,false);
SAPGuiPropIdStr("wnd[0]/tbar[0]/okcd");
SAPGuiCmdl(GuiOkCodeField,PutText,"bibs");
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmdl(GuiMainWindow,SendVKey,0);
SAPGuiCheckScreen( "SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access" );

SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:0200/subSA_200_1:SAPLEXAMP
LE_ENTRY_SCREEN:0800/cntlCC_HTML_INDEX/shellcont/shell");
SAPGuiCmd3(GuiCtrlHTMLViewer,SapEvent,"","","sapevent:ALV_SHORT?ALV");
SAPGuiCheckScreen( "BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Check boxes" );
```

SAPGuiCheckStatusBar

Specifies the method or property that is called or set, allowing the script access to the SAP GuiStatusBar object.

Syntax

```
SAPGuiCheckStatusBar (ID, statusBarValue);
```

Parameters


Parameter	Description
ID	ID to specify access to the status bar object.
statusBarValue	Status bar string to check against.

Example

```
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");
```

SAPGuiCmd0

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 0 in the name indicates that zero parameters are sent.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd0(Type, FuncName);
```

Parameters

Parameter	Description
Type	Type of object.
FuncName	Function or method/property related to the object.

Example


```
SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:
0200/subSA_200_2:SAPLEXAMPLE_ENTRY_SCREEN:
2000/cntlCCCONTAINER/shellcont/shell");
SAPGuiCmd2(GuiCtrlGridView, SetCurrentCell, -1, "SEATSOCC");
//Call GuiCtrlGridView class method ClearSelection
SAPGuiCmd0(GuiCtrlGridView, ClearSelection);
SAPGuiCreateColl(GuiCollection, CreateGuiCollection, coll1);
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "PRICE");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSMAX");
```

```
SAPGuiCmdlColl(GuiCollection, Add, coll1, "SEATSOCC");
SAPGuiCmdl(GuiCtrlGridView, PutSelectedColumns, coll1);
SAPGuiDestroyColl(GuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[3]");
SAPGuiCmd0(GuiButton, Press);
SAPGuiCheckScreen( "BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Alv grid" );
```

SAPGuiCmd1

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 1 in the name indicates that one parameter is sent.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd1(Type, FuncName, Param);
```

Parameters

Parameter	Description
Type	Type of object.
FuncName	Function or method/property related to the object.
Param	The first parameter to send.

Example

```
SAPGuiCmd1(GuiPasswordField, PutCaretPosition, 3);
SAPGuiCmd1Pwd(GuiPasswordField, PutText, "~encr~1111111111");


//This variation is to be used for entering passwords only.
SAPGuiCmd1Pwd(GuiPasswordField, PutText, "~encr~1111111111" );

//Call the GuiMainWindow class method SendVKey with one parameter that has a value of 0
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow, SendVKey, 0);
SAPGuiCheckScreen( "SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access" );
```

SAPGuiCmd1Coll

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 1 in the name indicates that one parameter is sent.

Use this variation to deal with Collection object information only.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd1Coll(Type, FuncName, Coll, Param);
```

Parameters

Parameter	Description
Type	Type of object.

FuncName	Function or method/property related to the object.
Coll	Collection name of collection.
Param	The first parameter to send.

Example

```
//multiple selections of columns
SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:
                0200/subSA_200_2:SAPLEXAMPLE_ENTRY_SCREEN:
                2000/cntlCCCONTAINER/shellcont/shell");
SAPGuiCmd2(GuiCtrlGridView,SetCurrentCell,-1,"SEATSOCC");
SAPGuiCmd0(GuiCtrlGridView,ClearSelection);
SAPGuiCreateColl(GuiCollection,CreateGuiCollection,coll1);


//adds columns to a collection that was created by the selection of columns
SAPGuiCmd1Coll(GuiCollection,Add,coll1, "PRICE");
SAPGuiCmd1Coll(GuiCollection,Add,coll1, "SEATSMAX");
SAPGuiCmd1Coll(GuiCollection,Add,coll1, "SEATSOCC");
SAPGuiCmd1(GuiCtrlGridView,PutSelectedColumns,coll1);
SAPGuiDestroyColl(GuiCollection,coll1);

SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[3]");
SAPGuiCmd0(GuiButton,Press);
SAPGuiCheckScreen("BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Alv grid" );
```

SAPGuiCmd1Elmnt

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 1 in the name indicates that one parameter is sent.

Use this variation for entering COM array element information only (VB collections).

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd1Elmnt(Type, SubPropType, FuncName, ElmntAry, ElmntIndx, SubFuncName, Param);
```

Parameters

Parameter	Description
Type	Type of object.
SubPropType	The type of the sub-property.
FuncName	Function or method/property related to the object.
ElmntAry	Name of the COM element array.
ElmntIndx	Index of location in array.
SubFuncName	Function name in collection array.
Param	The first parameter to send.

Example


```
SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:
    0200/subSA_200_2:SAPLEXAMPLE_ENTRY_SCREEN:
    2100/tblSAPLEXAMPLE_ENTRY_SCREENTC535");
SAPGuiCmd1(GuiTableControl, ReorderTable, "0 2 5 3 1 4 6 7" );

//Call GuiTableControl class of type
//GuiCollection with the GetColumns method.
//At elements 0, 4, and 2, set
//the width to 8, 8, and 7, respectively.
SAPGuiCmd1Elmnt(GuiTableControl, GuiCollection, GetColumns, ElementAt, 0, PutWidth, 8 );
SAPGuiCmd1Elmnt(GuiTableControl, GuiCollection, GetColumns, ElementAt, 4, PutWidth, 8 );
SAPGuiCmd1Elmnt(GuiTableControl, GuiCollection, GetColumns, ElementAt, 2, PutWidth, 7 );
```

SAPGuiCmd1Sub

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 1 in the name indicates that one parameter is sent.

Use this variation of the SAPGuiCmd command for dealing with subtype information only.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd1Sub(Type, SubPropType, FuncName, SubFuncName, Param);
```

Parameters

Parameter	Description
Type	Type of object.
SubPropType	The type of the sub-property.
FuncName	Function or method/property related to the object.
SubFuncName	Function name in the collection array.
Param	The first parameter to be sent.


Example

```
// Call GuiTableControl class of type
// GuiCollection. Get all columns and
// set the width to a value of 2.
SAPGuiPropIdStr("wnd[0]/usr/tblMP400100TC3000");
SAPGuiCmd1Sub1(GuiTableControl, GuiTableRow, GetAbsoluteRow, PutSelected, true, 0);
SAPGuiCmd1Sub(GuiTableControl, GuiCollection, GetColumns, ElementAt, -1, PutWidth, 2);
```

SAPGuiCmd1Sub1

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 1 in the name indicates that one parameter is sent.

Use this variation of the SAPGuiCmd command to deal with subtype and SubParameter information only.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd1Sub1(Type, SubPropType, FuncName, SubFuncName, Param, SubParam);
```

Parameters

Parameter	Description
Type	Type of object.
SubPropType	The type of the sub-property.
FuncName	Function or method/property related to the object.
SubFuncName	Function name in the collection array.
Param	The first parameter to send.
SubParam	The parameter to be sent to the SubFunction.


Example

```
//Call GuiTableControl class of type
//GuiTableRow. Call GetAbsoluteRow with
//a value of 0 and put a value of True.

SAPGuiPropIdStr("wnd[0]/usr/tblMP400100TC3000");
SAPGuiCmd1Sub1(GuiTableControl, GuiTableRow, GetAbsoluteRow, PutSelected, true, 0);
SAPGuiCmd1Sub(GuiTableControl, GuiCollection, GetColumns, ElementAt, -1, PutWidth, 2);
```

SAPGuiCmd2

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. The 2 in the name indicates that two parameters are sent.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCmd2(Type, FuncName, Param1, Param2);
```

Parameters

Parameter	Description
Type	Type of object.
FuncName	Function or method/property related to the object.
Param1	The first parameter to send.
Param2	The second parameter to send.

Example

```
//Call SetCurrentCell with two parameters

SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:
                0200/subSA_200_2:SAPLEXAMPLE_ENTRY_SCREEN:
                2000/cnt1CCCONTAINER/shellcont/shell");
SAPGuiCmd2(GuiCtrlGridView, SetCurrentCell, -1, "SEATSOCC");
SAPGuiCmd0(GuiCtrlGridView, ClearSelection);
```


QALoad 5.02

```
SAPGuiCreateColl(GuiCollection, CreateGuiCollection, coll1);
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "PRICE");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSMAX");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSOCC");
SAPGuiCmd1(GuiCtrlGridView, PutSelectedColumns, coll1);
SAPGuiDestroyColl(GuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[3]");
SAPGuiCmd0(GuiButton, Press);
SAPGuiCheckScreen( "BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Alv grid" );
```

SAPGuiCmd3

Specifies the method or property that is called or set in the object specified in the previous `SAGuiPropIDStr` call. The 3 in the name indicates that three parameters are sent.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "*SAP GUI Scripting API for the Windows and Java Platforms*".

Syntax

```
SAPGuiCmd3(Type, FuncName, Param1, Param2, Param3);
```

Parameters

Parameter	Description
Type	Type of object.
FuncName	Function or method/property that is related to the object.
Param1	The first parameter to send.
Param2	The second parameter to send.
Param3	The third parameter to send.

Example

```
//Resize the main window
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd3(GuiMainWindow, ResizeWorkingPane, 94, 24, false);
```

SAPGuiConnect

Specifies to which server description a connection should be made.

Syntax

```
SAPGuiConnect( s_info, server description);
```

Parameters


Parameter	Description
s_info	Structure used by each virtual user.
server description	String that matches a server in the SAPLogon specifications.

Example

```
//Connect to the SAP server named testsap620
SAPGuiConnect( s_info, "testsap620");
SAPGuiVerCheckStr( "6205.132.36");
```

SAPGuiCreateColl

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. This call creates a collection with the name specified by Coll.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiCreateColl(Type, FuncName, Coll)
```

Parameters

Parameter	Description
Type	Type of object.
FuncName	Function or method/property related to the object.
Coll	Collection name of collection.

Example

```
SAPGuiCreateColl(GuiCollection, CreateGuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:
    SAPLEXAMPLE_ENTRY_SCREEN:0200/subSA_200_2:
    SAPLEXAMPLE_ENTRY_SCREEN:2000/cntlCCCONTAINER/shellcont/shell");


SAPGuiCmd2(GuiCtrlGridView, SetCurrentCell, -1, "SEATSOCC");
SAPGuiCmd0(GuiCtrlGridView, ClearSelection);

//Multiple selections of columns creates a collection for the selection of columns
SAPGuiCreateColl(GuiCollection, CreateGuiCollection, coll1);
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "PRICE");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSMAX");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSOCC");
SAPGuiCmd1(GuiCtrlGridView, PutSelectedColumns, coll1);
SAPGuiDestroyColl(GuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[3]");
SAPGuiCmd0(GuiButton, Press);
SAPGuiCheckScreen( "BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Alv grid" );
```

SAPGuiDestroyColl

Specifies the method or property that is called or set in the object specified in the previous SAPGuiPropIdStr call. This call destroys a collection and decrements reference count.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiDestroyColl(Type, Coll);
```

Parameters

Parameter	Description
Type	Type of object.
Coll	Collection name of collection

Example

```
SAPGuiDestroyColl(GuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/usr/subSA_0100_1:SAPLEXAMPLE_ENTRY_SCREEN:
                0200/subSA_200_2:SAPLEXAMPLE_ENTRY_SCREEN:
                2000/cntlCCCONTAINER/shellcont/shell");
SAPGuiCmd2(GuiCtrlGridView, SetCurrentCell, -1, "SEATSOCC");
SAPGuiCmd0(GuiCtrlGridView, ClearSelection);
SAPGuiCreateColl(GuiCollection, CreateGuiCollection, coll1);
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "PRICE");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSMAX");
SAPGuiCmd1Coll(GuiCollection, Add, coll1, "SEATSOCC");
SAPGuiCmd1(GuiCtrlGridView, PutSelectedColumns, coll1);
//Multiple selections of columns
//destroys a collection that was
//created by a selection of columns
SAPGuiDestroyColl(GuiCollection, coll1);

SAPGuiPropIdStr("wnd[0]/tbar[0]/btn[3]");
SAPGuiCmd0(GuiButton, Press);
SAPGuiCheckScreen( "BIBS", "SAPLEXAMPLE_ENTRY_SCREEN", "Style Guide: Alv grid" );
```

SAPGuiPropIdStr

Specifies the object ID string to use with subsequent SAPGui calls. This object ID remains in effect until another call to SAPGuiPropIdStr is made.

Syntax

```
SAPGuiPropIdStr(Object Id);
```

Parameters

Parameter	Description
Object ID	String used for subsequent SAPGui command calls.

Example

```
//Set the object ID to "wnd[0]"
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(GuiMainWindow, SendVKey, 0);
SAPGuiCheckScreen( "SESSION_MANAGER", "SAPLSMTR_NAVIGATION", "SAP Easy Access" );
```

SAPGuiPropIdStrExists

Specifies the object ID string to use with subsequent SAPGui calls. This object ID remains in effect until another call to SAPGuiPropIdStr or SAPGuiPropIdStrExists is made.

Syntax

```
SAPGuiPropIdStrExists (char* Object_Id);
```

Parameters

Parameter	Description
Object_Id	String used for subsequent SAPGui command calls.

Example

```
SAPGuiCheckScreen("S000", "SAPMSYST", "SAP");
DO_SLEEP(3);
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");
    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");
```

SAPGuiPropIdStrExistsEnd

Marks the end of the block of code that is executed if the condition in a prior `SAPGuiPropIdStrExists` command is true.

Syntax

```
SAPGuiPropIdStrExistsEnd (char* Object_Id);
```

Parameters


Parameter	Description
Object_Id	String used for matching <code>SAPGuiPropIdStrExists</code> command calls.

Example

```
SAPGuiCheckScreen("S000", "SAPMSYST", "SAP");
DO_SLEEP(3);
SAPGuiPropIdStrExists("wnd[1]/usr/radMULTI_LOGON_OPT2");
    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");
```

SAPGuiSessionInfo

Specifies the method or property that is called allowing the script access to the SAP `GuiSessionInfo` objects.

 **Note:** For more information about SAP parameters, refer to SAP's publication titled "SAP GUI Scripting API for the Windows and Java Platforms".

Syntax

```
SAPGuiSessionInfo(FuncName, Param1)
```

Parameters

Parameter	Description
FuncName	Function or method/property related to the object.
Param1	The first parameter to send.

Example

In this example, RoundTrip data and Flush data are stored in custom counters

```
int id1, id2, id3, id4;
long lRoundTrips, lFlushes;

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);

.
.
.

//Retrieve the number of round trips
SAPGuiSessionInfo(GetRoundTrips, lRoundTrips);
//Retrieve the number of times the buffer is flushed
SAPGuiSessionInfo(GetFlushes, lFlushes);
SAPGuiPropIdStr("wnd[1]/usr/btnSPOP-OPTION1");
SAPGuiCmd0(GuiButton, Press);
SAPGuiCheckScreen("SESSION_MANAGER", "SAPLSP01", "Log Off" );

COUNTER_VALUE(id1, lRoundTrips);
COUNTER_VALUE(id2, lFlushes);
COUNTER_VALUE(id3, lRoundTrips);
COUNTER_VALUE(id4, lFlushes);
```

SAPGuiSetCheckScreenWildcard

Specifies the wildcard character that SAPGuiCheckScreen uses for wildcard matching.

Although all converted scripts include SAPGuiSetCheckScreenWildcard ('*') as one of the first functions, you can specify a different wildcard character to use later in the script.

Syntax

```
SAPGuiSetCheckScreenWildcard (unsigned short wildcard);
```

Parameters

Parameter	Description
wildcard	A character to match in SAPGuiCheckScreen.

Example

```
//Set the wildcard character to *
SAPGuiSetCheckScreenWildcard('*');
SYNCHRONIZE();
BEGIN_TRANSACTION();
```

```

try{
SAPGuiConnect( s_info, "testsap620");
SAPGuiVerCheckStr( "6205.132.36" );

.
.
.
}
catch(_com_error e){

.
.
.
}

```

SAPGuiVerCheckStr

Specifies the SAP GUI front end version number at the time the capture file was made.

This information includes the major version, the minor version, and the patch level that was installed at the time of capture. If the information does not match, it may not be possible to do a playback from this capture.

Syntax

```
SAPGuiVerCheckStr("Major version.Minor version.Patchlevel");
```

Parameters

Parameter	Description
Major version.Minor version.Patchlevel	String that includes the major version number, minor version number, and patch level number of the installed SAP client.

Example

```

SAPGuiConnect(s_info, "testsap620");
SAPGuiVerCheckStr( "6204.119.32" );

```

SSL

SSL Index

DO_Https

Applies to **SSL** requests. Makes a secured request to the server specified by the `http_statement`.

DO_SetSSLConnectString

Applies to **SSL** requests. Sets the proxy authorization when accessing **SSL** pages passed through a proxy server (also known as "SSL tunneling").

DO_SSLReuseSession

Applies to **SSL** requests. Re-uses the current session's communication information (session ID) for all page requests within the transaction.

DO_SSLUseCipher

Applies to **SSL** requests. Sets the encryption algorithm for playback.

DO_SSLUseClientCert

Applies to SSL requests. Specifies a client certificate to pass upon request while recording SSL requests.

DO_SSLUseClientCertPass

Applies to SSL requests. Specifies a password (plain text or encrypted) that is needed to read a client certificate.

DO_SSLUseProxy

Applies to SSL requests. Specifies a proxy server for all SSL requests to be sent through.

DO_Https

Applies to SSL requests. Makes a secured request to the server specified by the http_statement.

This command returns a string containing the HTML response from the secured server.

Syntax

```
DO_Https ( const char *http_statement );
```

Parameters

Parameter	Description
http_statement	A string containing the URL of the secured server and any headers to be sent.


Example

```
...
...
DO_Https("GET HTTPS://www.yahoo.com HTTP/1.0\r\n"
        "Referer: HTTP://company/index.htm\r\n"
        "Proxy-Connection: Keep-Alive\r\n"
        "User-Agent: Mozilla/3.01 WinNT;I)\r\n"
        "Host: www.yahoo.com\r\n"
        "Accept: /**\r\n");
...
...
```

DO_SetSSLConnectString

Applies to SSL requests. Sets the proxy authorization when accessing SSL pages passed through a proxy server (also known as "SSL tunneling").

This command will be called for each SSL request connecting to a different server.

 **Note:** DO_SetSSLConnectString is a deprecated command. It is used internally by QALoad . Connection strings are created internally by QALoad . In addition, the DO_SetSSLConnectString command will be commented out in converted scripts to help create a custom connect string if needed.

Syntax

```
int DO_SetSSLConnectString ( const char *connectstring ) ;
```

Parameters

Parameter	Description
-----------	-------------

connectstring	A character string specifying the command to be sent to the SSL proxy server to allow SSL requests to be sent. This is in the format "CONNECT servername:port". The connect string must be terminated by a double CR-LF pair.
---------------	---

Example

```

...
...
DO_SetSSLConnectString("CONNECT www.yahoo.com:443 HTTP/ 1.0\n"
"Proxy-authorization: Basic cGZobGFwMDpicm9uaWNh\r\n"
"User-Agent: Mozilla/4.04 [en] (WinNT; U)\r\n\r\n");
DO_Https("GET HTTPS://www.yahoo.com HTTP/1.0\r\n"
"Referer: HTTP://company/index.htm\r\n"
"Proxy-Connection: Keep-Alive\r\n"
"User-Agent: Mozilla/3.01 WinNT;I)\r\n"
"Host: www.yahoo.com\r\n"
"Accept: */*\r\n");
...
...

```

DO_SSLReuseSession

Applies to SSL requests. Re-uses the current session's communication information (session ID) for all page requests within the transaction.

DO_SSL_ReuseSession is related to the option Reuse SSL Session ID check box on the WWW Advanced dialog box. The WWW Advanced dialog box is accessed from the Convert Options wizard by clicking the Advanced button.

Place DO_SSLReuseSession before the BEGIN_TRANSACTION statement to use the session ID for all transactions, or place it after the BEGIN_TRANSACTION statement to reuse the session ID only for statements within that transaction.

Syntax

```
DO_SSLReuseSession( BOOL bEnable );
```

Parameters

Parameter	Description
bEnable	Starts (TRUE) or stops (FALSE) the reuse of a session ID.

Examples

In the following example, the very first SSL connection will establish a Session ID, which will be reused again for all SSL requests and transactions accessing the same Web server:

```

...
...
DO_SSLReuseSession(1);
BEGIN_TRANSACTION();
...
...
END_TRANSACTION();
...
...

```

In the following example, the first SSL connection within a transaction will establish a Session ID, which will be reused again for all SSL requests accessing the same Web Server within the same transaction:


```
...
...
BEGIN_TRANSACTION();
DO_SSLReuseSession(1);
...
...
END_TRANSACTION();
...
...
```

DO_SSLUseCipher

Applies to SSL requests. Sets the encryption algorithm for playback.

By default, QALoad scripts negotiate the strongest common SSL cipher for each SSL session. The Convert facility automatically inserts a commented out DO_SSLUseCipher whenever it encounters an encryption algorithm that changed while recording. You can uncomment this call to force playback to use a specific cipher.

It is possible to change the algorithms, and even choose to have several encryption algorithms in one script.

 **Note:** DO_SSLUseCipher is a deprecated command. Cipher selection is done internally by QALoad. If you do not have an encryption license, the listed encryption codes will not work. If you have an export grade license, only 40-bit codes will work with your scripts; however, if you have a 128-bit license, all of the listed codes will work with your scripts.

Encryption Algorithms

The codes for available algorithms are as follows:

Export grade (40 bit):

```
EXP-EDH-RSA-DES-CBC
EXP-EDH-DSS-DES-CBC-SHA
EXP-DES-CBC-SHA
EXP-RC4-MD5
EXP-RC2-CBC-MD5
```

128-bit encryption:

```
RC4-SHA
RC4-MD5
EDH-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
DES-CBC3-SHA
EDH-RSA-DES-CBC-SHA
EDH-DSS-DES-CBC-SHA
DES-CBC-SHA
DES-CBC3-MD5
DES-CBC-MD5
RC2-CBC-MD5
```

Syntax

```
int DO_SSLUseCipher(const char *cipher)
```


Parameters

Parameter	Description
cipher	A character string representing the encryption algorithm to be used during playback.

Example

```

...
...
BEGIN_TRANSACTION();
...
...
DO_SSLUseCipher("EXP-RC4-MD5");
...
...
END_TRANSACTION();
...
...

```

DO_SSLUseClientCert

Applies to **SSL** requests. Specifies a client certificate to pass upon request while recording **SSL** requests.

QALoad's convert facility will use the name of the certificate used while recording. The certificate can be selected from the QALoad Script Development Workbench Record Options wizard.

Syntax

```
int DO_SSLUseClientCert(const char *name);
```

Parameters

Parameter	Description
name	A string containing the name of the client certificate to use.

Example

In the following example, the client certificate "qaload_cl" will be used whenever the server requests one.

```
DO_SSLUseClientCert("qaload_cl");
```

DO_SSLUseClientCert

Applies to **SSL** requests. Specifies a password (plain text or encrypted) that is needed to read a client certificate.

Syntax

```
BOOL DO_SSLUseClientCertPass(const char *szPassword);
```

Parameters

Parameter	Description
szPassword	A string containing the password to use.

Example

```
DO_SSLUseClientCert( "my_passwd" );
```

DO_SSLUseProxy

Applies to SSL requests. Specifies a proxy server for all SSL requests to be sent through.

Syntax

```
int DO_SSLUseProxy ( const char *proxyURL ) ;
```

Parameters

Parameter	Description
proxyURL	A character string indicating the servername and port of the proxy server, specified in "servername:port" format.

Example

```
...
...
BEGIN_TRANSACTION();
...
...
DO_UseProxy ( "internet.company.com:80" );
DO_SSLUseProxy ( "internet.company.com:90" );
DO_ProxyExceptions( "company.sample.com, "company2.company.com" );
...
...
```

Tuxedo

Tuxedo Index

Do_TuxAppendBuffer

Used in conjunction with Do_TuxBuildBuffer() to build large Tuxedo buffers and overcome internal compiler limitations.

Do_TuxBufMemset

Provides a way to directly replace the offset of a Tuxedo buffer with the contents of the parameter Data.

Do_TuxBuildBuffer

If the application is sending very large amounts of data to the server within a single buffer variable, it is possible that string constants used within the script can cause compiler errors just by their shared size. This command, along with Do_TuxAppendBuffer() allows the Convert facility to break up large strings into smaller ones. These strings are then appended at runtime to build the contents of a large Tuxedo buffer.

Do_Tuxcarray

Loads the specified data into a tallocated buffer of type Carray.

Do_Tuxcertsubstr

Substitutes the recorded certificate with a valid certificate at replay time.

Do_Tuxencode

Converts a character array into an encoded string.

Do_TuxFinit

Determines the FML type being used and initializes an FML or FML32 buffer, based on how the buffer was allocated.

Do_TuxFMLData

Adds data to an existing 16- or 32-bit FML buffer. The `buffer_idx` value from the most recent `Do_TuxFinit` command determines the buffer.

Do_TuxgetFMLData

Retrieves data from an FML or FML32 buffer for subsequent use within the script.

Do_TuxGetPartialBuffer

Gets a pointer to partial buffer built with `Do_TuxBuildBuffer`.

Do_Tuxgetrevent

The `revent` parameter is used in `tpsend` and `tprecv` to indicate whether the `send/receive` was successful. `Revent` is set after each `tpsend` and `tprecv` command.

Do_TuxgetTuxBuffer

Use this command to directly access an FML buffer or any other Tuxedo buffer type.

DO_TuxInitData

Initializes QALoad 's internal data arrays and buffers prior to calling all other `DO_Tux` commands.

Do_TuxOutputBuffer

Provides a way of outputting a `CARRAY`, `X_OCTET`, or `STRING` buffer type to the player window.

Do_TuxSetViewData

`DO_TuxSetViewData` converts an encoded string from data into raw bytes, and copies it into a view buffer variable, then returns the number of actual bytes stored. This command is used to set view data of type `string` and `carry`.

Do_TuxSetViewEnv

Sets the internal Tuxedo environment variables, `VIEWFILES`, `VIEWDIR`, `VIEWFILES32`, and `VIEWDIR32`.

Do_Tuxsetwsnaddr

Sets the `WSNADDR` parameter for the session and overrides any `WSNADDR` parameter that might be in the current environment.

Do_Tuxstring

Loads the specified data into a `tpallocated` buffer of type `char *`. If the buffer is too small to hold the data, `Do_Tuxstring` returns an error message at replay.

Do_TuxStrlenEncodeString

Encodes a string such that the resulting string is the concatenation of the string length (encoded) of the input string followed by the string itself.

Do_Tuxsubstr

Provides a way to replace all strings with another string within a specified buffer index.

Do_Tuxtpabort

Aborts an active Tuxedo transaction.

Do_Tuxtpalloc

QALoad 5.02

Calls a Tuxedo tppalloc command and stores the resultant address in a buffer table.

[Do_Tuxtbegin](#)

Calls a Tuxedo tbegin command.

[Do_Tuxtpbroadcast](#)

Calls a Tuxedo tpbroadcast command.

[Do_Tuxtpcall](#)

Calls a Tuxedo tpcall command. Prior to making the tpcall, it clears QALoad 's timer.

[Do_Tuxtpcommit](#)

Calls a Tuxedo tpcommit command.

[Do_Tuxtpconnect](#)

Connects the client application to a conversational server.

[Do_Tuxtpdiscon](#)

Disconnects the client application from a conversational server.

[Do_Tuxtpenqueue](#)

Enqueues a message to a Tuxedo queue. Prior to calling the Tuxedo command tpenqueue, Do_Tuxtpenqueue clears QALoad 's timer.

[Do_Tuxtpinit](#)

Calls a Tuxedo tpinit command.

[Do_Tuxtpost](#)

Posts a Tuxedo event.

[Do_Tuxtprealloc](#)

Calls a Tuxedo tprealloc command to resize a previously allocated Tuxedo buffer.

[Do_Tuxtprecv](#)

Calls a Tuxedo tprecv command to receive data from a conversational server.

[Do_Tuxtpscmt](#)

Calls the Tuxedo command tpscmt. In turn, tpscmt sets the TP_COMMIT_CONTROL characteristic to the value specified in flags.

[Do_Tuxtpsend](#)

Calls a Tuxedo tpsend command which sends data to a conversational server.

[Do_Tuxtpsprio](#)

Calls the Tuxedo tpsprio command which sets the priority for the next request sent.

[Do_Tuxtpterm](#)

Terminates QALoad 's connection to the Tuxedo server.

[Do_TuxUseCertificates](#)

Used by the system to set the certificates flag when using Peoplesoft certificates. See Do_Tuxcertsubstr for more information.

[Do_Tuxxoctet](#)

Loads the specified data into a tpallocated buffer of type X_OCTET.

Do_TuxAppendBuffer

Used in conjunction with Do_TuxBuildBuffer() to build large Tuxedo buffers and overcome internal compiler limitations.

Please refer to the documentation for Do_TuxBuildBuffer() for additional explanation.

Syntax

```
void Do_TuxAppendBuffer( char * data );
```

Parameters

Parameter	Description
data char *	Encoded string to be appended to the internal buffer.

Example

```
Do_TuxBuildBuffer( 64000, "&01&02&03&04&05");
Do_TuxAppendBuffer( "&89&8a&8b&8c" );
Do_Tuxcarray( Buf2, Do_TuxGetPartialBuffer() );
```

Do_TuxBufMemset

Provides a way to directly replace the offset of a Tuxedo buffer with the contents of the parameter Data. This function will not write to an area outside the Tuxedo buffer passed in the parameter Index.

Syntax

```
void Do_TuxBufMemset( int Index, int Offset, char * Data );
```

Parameters


Parameter	Description
Index integer	Index into the buffer table.
Offset integer	Offset into the buffer.
Data char *	Source string to overwrite with.

Example

```
Do_Tuxtpalloc( Buf1, "CARRAY", 1024 );
Do_Tuxcarray( Buf1, "This is test data." );
Do_TuxBufMemset( Buf1, 8, "real" );
Do_TuxOutputBuffer( Buf1, 0, 20, 80 );
```

Output:

```
"[ 0]"This is real data.~~"
```

 **Note:** [xxxxx] precedes each output line to assist you in determining the current offset position in the buffer.

Do_TuxBuildBuffer

If the application is sending very large amounts of data to the server within a single buffer variable, it is possible that string constants used within the script can cause compiler errors just by their shared size. This command, along with Do_TuxAppendBuffer() allows the Convert facility to break up large strings into smaller ones. These strings are then appended at runtime to build the contents of a large Tuxedo buffer.

Call Do_TuxBuildBuffer() to initialize the large buffer. Previous contents of the buffer are erased.

Syntax

```
void Do_TuxBuildBuffer( int length, char * data );
```

Parameters

Parameter	Description
length integer	Maximum length of the buffer.
data char *	Encoded string to be copied to the buffer. The data is copied to the buffer in the encoded format.

Example

```
Do_TuxBuildBuffer( 64000, "&01&02&03&04&05");
Do_TuxAppendBuffer( "&89&8a&8b&8c" );
Do_Tuxcarray( Buf2, Do_TuxGetPartialBuffer() );
```

Do_Tuxcarray

Loads the specified data into a tallocated buffer of type Carray.

If the buffer is not large enough to hold the data, you will receive an error message.

Syntax

```
void Do_Tuxcarray( int buffer_idx, char * data );
```

Parameters

Parameter	Description
buffer_idx integer	Index into the buffer table.
data char *	Pointer to an encoded string containing the data to place into the buffer.

Example

```
Do_Tuxtpalloc( 3, "CARRAY", 3200 );
Do_Tuxcarray( 3, "~&01&02&03"; )
```

Do_Tuxcertsubstr

Substitutes the recorded certificate with a valid certificate at replay time.

This function is rendered at convert time, and normally does not have to be modified/ added to the resulting script. The recorded and replay strings are required to be encoded, and of length 16 characters (binary).

Syntax

```
int Do_Tuxcertsubstr ( int Index, char * Src, char * Dest );
```

Parameters

Parameter	Description
Index int	Index into the buffer table.
Src char *	Pointer of string to search for (capture certificate).
Dest char *	Pointer of value to replace (replay certificate).

Returns

Int: 1 if substitution is made, 0 if substitution is not made.

Example

```
...
char captured_certificate[48];
char replay_certificate[48];
...
...
Do_Tuxtpcall ( "GetCertificate", Buf1, Buf2, TPSIGRSTRT );
if ( USE_CERTIFICATES )
{
memcpy( replay_certificate, tuxBuffer[Buf2].address + 35, 16 );
Do_Tuxcarray( Buf2, "x&ce&c4&b8&fd&b3&e7&a0&ee&a 9&fc&b1&f4&ba&ee&a9");
memcpy( captured_certificate, tuxBuffer [Buf2].address, 16 );
}
...
Do_Tuxcarray ( Buf1, "&ae~~~&04&03&02&01&01~~~&bc&02~~~~~"
"m~~~&04~SCTX&08NGUYENT2&08NGUYENTG&08NGUYENTG"
"x&ce&c4&b8&fd&b3&e7&a0&ee&a9&fc&b1&f4&ba&ee&a9~~~"
"~~~~~&03ENG~~~~~&01$&01/&01:&06M/d/yy"
"&01.&01,&02AM&02PM~~~~ &09~SClearReq~~~~~"
"&ff&fd&01~~");
Do_Tuxcertsubstr ( Buf1, captured_certificate, replay_certificate );
Do_Tuxtpcall ( "MgrClear", Buf1, Buf2, TPSIGRSTRT );
```

Do_Tuxencode

Converts a character array into an encoded string.

Syntax

```
void Do_Tuxencode( char * dest, char * Src, long SrcLen );
```

Parameters

Parameter	Description
dest char *	Pointer to a buffer which receives the encoded string. Make sure that the destination buffer is large enough to contain the encoded string.
Src char *	Pointer to a buffer containing the source data. This data may contain NULLs and other non-printable characters.
SrcLen long	Number of bytes in the source buffer. If SrcLen is -1, then Src is treated as a NULL terminated string. The final NULL character is not included in the

encoded string.

Example

```
char dest[30];
char src[]="\0\xff\xfe\xfd"
Do_Tuxencode( dest, src, 10 );
```

Do_TuxFinit

Determines the FML type being used and initializes an FML or FML32 buffer, based on how the buffer was allocated.

Do_TuxFinit stores the `buffer_idx` value in a global variable, so it does not have to be specified in the Do_TuxFMLData command.

Syntax

```
void Do_TuxFinit( int buffer_idx );
```

Parameters

Parameter	Description
<code>buffer_idx</code> integer	Index into the buffer table.

Example

```
Do_TuxFinit( 1 ); /* For: tpcall */
Do_TuxFMLData( test_short, 0, "-12345" );
Do_TuxFMLData( test_short, 1, "0" );
Do_TuxFMLData( test_short, 2, "32767" );
Do_Tuxtpcall( "OPEN_TEST1", 1, 1, 0 );
```

Do_TuxFMLData

Adds data to an existing 16- or 32-bit FML buffer. The `buffer_idx` value from the most recent Do_TuxFinit command determines the buffer.

The data parameter — all non-printable characters as well as some special characters — is encoded as a three-byte hex sequence. Do_TuxFMLData first converts the encoded string into an internal ASCII form and then moves it into the Tuxedo FML buffer.

Syntax

```
void Do_TuxFMLData( int field, int occ, char * data );
```

Parameters

Parameter	Description
<code>field</code> integer	FML field value or symbolic constant.
<code>occ</code> integer	FML field occurrence.
<code>data</code> char *	Pointer to an encoded string containing the data to place into the buffer.

Example

```
Do_TuxFinit( 1 ); /* For: tpcall */
Do_TuxFMLData( test_short, 0, "-12345" );
Do_TuxFMLData( test_short, 1, "0" );

/* encoded string*/
Do_TuxFMLData( test_string, 0, "abc&01&fe" );
Do_Tuxtpcall( "OPEN_TEST1", 1, 1, 0 );
```

Do_TuxgetFMLData

Retrieves data from an FML or FML32 buffer for subsequent use within the script.

The system automatically determines which type of FML commands to use (FML or FML32) based upon the type of the buffer.

Before calling Do_TuxgetFMLData, you must allocate enough storage for returned data.

The char * that Do_TuxgetFMLData returns to the data area helps facilitate Do_TuxgetFMLData's use in commands such as strcpy and printf.

This command does not perform any conversion on the original data type; therefore, you must know how the data is stored before using the returned value.

Syntax

```
char * Do_TuxgetFMLData( int buffer_idx, long fieldID, long occ, char * data );
```

Parameters

Parameter	Description
buffer_idx integer	Index into the buffer table.
fieldID long	Symbolic constant for numeric value for the desired field.
occ long	Field occurrence.
data char *	Pointer to the buffer that receives the data.

Returns

char *: String data

Example

```
char data[128];
printf("Account: %s\n", Do_TuxgetFMLData(1, ACCT_ID, 0, data) );
```

Do_TuxGetPartialBuffer

Gets a pointer to partial buffer built with Do_TuxBuildBuffer.

Syntax

```
char * Do_TuxGetPartialBuffer();
```

Parameters

None.

Returns

char *: Encoded string data

Example

```
Do_TuxBuildBuffer( 64000, "&01&02&03&04&05");
Do_TuxAppendBuffer( "&89&8a&8b&8c" );
Do_Tuxcarray( Buf2, Do_TuxGetPartialBuffer() );
```

Do_Tuxgetrevent

The revent parameter is used in tpsend and tprecv to indicate whether the send/receive was successful. Revent is set after each tpsend and tprecv command.

Syntax

```
int Do_Tuxgetrevent( int conn_index );
```

Parameters

Parameter	Description
conn_index int	Index into a table of connection descriptors. The first descriptor is 1.

Returns


int: Current value of revent.

Example

```
if ( Do_Tuxgetrevent( 1 ) == TPEV_DISCONIMM )
printf( " Improper conversation disconnection." );
```

Do_TuxgetTuxBuffer

Use this command to directly access an FML buffer or any other Tuxedo buffer type.

 **Note:** Since the returned value points to a tallocated Tuxedo buffer, any subsequent Tuxedo calls can cause the buffer to be moved and/or reallocated.

Syntax

```
char * Do_TuxgetTuxBuffer( int buffer_idx, char * data );
```

Parameters

Parameter	Description
buffer_idx integer	Index into the buffer table.
data char *	Pointer to the buffer area.

Returns

char *: Pointer to the buffer area specified by buffer_idx.

Example

```
char *data; stringData[128], carrayData[2048];
/* Copy string data */
strcpy( stringData, Do_TuxgetTuxBuffer( 1, data ) );
```

```
/* Copy Carray data */
memcpy( carrayData, Do_TuxgetTuxBuffer( 2, data ) );
```

DO_TuxInitData

Initializes QALoad's internal data arrays and buffers prior to calling all other "DO_Tux" commands.

The QALoad convert facility usually determines the values for each of the parameters; however, the parameters may be overridden if you want to allocate additional Tuxedo buffers or connections.

Your script should only execute the Do_TuxInitData command once; therefore, place it before the BEGIN_TRANSACTION command.

Syntax

```
void DO_TuxInitData ( int bufCount, int conCount );
```

Parameters

Parameter	Description
bufCount int	Number of Tuxedo buffers to allocate. Maximum of 255.
conCount int	Number of connections to allocate. Maximum of 255.

Example

```
Do_TuxInitData ( 6, 0 );
SYNCHRONIZE( );
BEGIN_TRANSACTION( );
Do_Tuxsetwsnaddr( "//LUCKY:3107" );
Do_Tuxtpinit( "Smith", "bnkapp", "passwd", "", TPU_DIP, "" );
```

Do_TuxOutputBuffer

Provides a way of outputting a CARRAY, X_OCTET, or STRING buffer type to the player window.

Buffers of type FML or FML32 should use Do_TuxgetFMLData () to retrieve FML values. If the inCount value exceeds the size of the buffer, only the buffer contents will be printed. The offset of the encoded character precedes the output line to assist you in determining the current offset position in the buffer.

Syntax

```
int Do_TuxOutputBuffer ( int index, int offset, int inCount,
int printWidth );
```

Parameters

Parameter	Description
index int	Index into the buffer table.
offset int	Offset into buffer to start printing.
inCount int	Number of encoded characters to print.
printWidth int	Printed column width.

Returns


int: Printed column width.

Example

```
Do_Tuxtpalloc ( Buf1, "CARRAY", 8192 );
Do_Tuxcarray( Buf1, "This is ASCII text in a Tuxedo buffer."
"This is not ASCII text - &01&02&03&04&05~~~" );
Do_TuxOutputBuffer ( Buf1, 0, 100, 80 );
Do_TuxOutputBuffer ( Buf1, 44, 100, 80 );
Do_TuxOutputBuffer ( Buf1, 0, 44, 80 );
```

Output

```
[ 0]"This is ASCII text within a Tuxedo buffer. This is"
"not ASCII text - &01&02&03&0"
[ 80]"4&05~~~~~"
[ 44]"This is not ASCII test - &01&02&03&04&05"
"~~~~~"
[ 124]"~~~~~"
[ 0]"This is ASCII text within a Tuxedo buffer. "
```

 **Note:** "[xxxxx]" precedes each output line to assist you in determining the current offset position in the buffer.

Do_TuxSetViewData

Do_TuxSetViewData converts an encoded string from data into raw bytes, and copies it into a view buffer variable, then returns the number of actual bytes stored. This command is used to set view data of type string and carray.

Syntax

```
int DO_TuxSetViewData( char * field, char * data );
```

Parameters

Parameter	Description
field char *	Pointer to a view carray or string variable.
data char *	Encoded string to be converted and copied.

Returns

int: Number of bytes actually copied.

Example

```
Do_TuxSetViewData( VW_testVw16(Buf2)->tv16stringdata, "test data&13&10" );
/* Set length along with string data */
VW_testVw16Complex(Buf2)->L_tv16string2[0] = Do_TuxSetViewData( VW_testVw16Complex(Buf2) -
>tv16stringdata, "Hello world." );
```

Do_TuxSetViewEnv

Sets the internal Tuxedo environment variables, VIEWFILES, VIEWDIR, VIEWFILES32, and VIEWDIR32.

If this command is not present, then these environment variables, if needed, are expected to be passed by the operating system to the application. The values set in this command will override the system environment values only for this process.

Note: Windows-based directories with "\" characters in them should be entered as "\\ ", as shown in the example.

Syntax

```
void Do_TuxSetViewEnv( char * viewFiles, char * viewDir, char * viewFiles32, char * viewDir32 );
```

Parameters

Parameter	Description
viewFiles char *	Value for the VIEWFILES environment.
viewDir char *	Value for the VIEWDIR environment.
viewFiles32 char *	Value for the VIEWFILES32 environment.
viewDir32 char *	Value for the VIEWDIR32 environment.

Example

```
Do_TuxSetViewEnv( "testvw16.vv", "d:\\Tuxedo\\apps\\bankapp\\nt\\client", "testvw32.vv", "d:\\Tuxedo\\apps\\bankapp\\nt\\client" );
```

Do_Tuxsetwsnaddr

Sets the WSNADDR parameter for the session and overrides any WSNADDR parameter that might be in the current environment.

Syntax

```
void Do_Tuxsetwsnaddr( char * wsnaddr );
```

Parameters

Parameter	Description
wsnaddr char *	A valid WSNADDR string.

Example

```
Do_Tuxsetwsnaddr( "//LUCKY:3107" );
Do_Tuxtpinit( "Smith", "bnkapp", "mypass", "", TPU_DIP, "" );
```

Do_Tuxstring

Loads the specified data into a tallocated buffer of type char *. If the buffer is too small to hold the data, Do_Tuxstring returns an error message at replay.

Syntax

```
void Do_Tuxstring ( int buffer_idx, char * data );
```

Parameters

Parameter	Description
buffer_idx int	Index into the buffer table.
data char *	Pointer to an encoded string containing the data to place into the buffer.

Example

```
Do_Tuxtppalloc( 4, "STRING", 64 );
Do_Tuxstring( 4, "&01&02&03&04$%&26' ( ) *+, -. /01234567889:;<=" );
```

Do_TuxStrlenEncodeString

Encodes a string such that the resulting string is the concatenation of the string length (encoded) of the input string followed by the string itself.

For example, if the input string is John Smith, the resulting output of the Dest string would be &0aJohn Smith, where the length of John Smith equals 10, and 10 in hexadecimal equals 0a (encoded as &0a).

Syntax

```
int Do_TuxStrlenEncodeString( char * Src, char * Dest );
```

Parameters

Parameter	Description
Src char *	Pointer to string to encode.
Dest char *	Pointer to destination string.

Returns

int: String length of Dest.

Example

```
char temp[256];
char dest[256];
strcpy( temp, "Hello, world." );
printf( "Input String = %s\n", temp );
Do_TuxStrlenEncodeString( temp, dest );
printf( "Encoded String = %s\n", dest );
```

Output

```
"Input String = Hello, world."
"Encoded String = &0dHello, world."
```

Do_Tuxsubstr

Provides a way to replace all strings with another string within a specified buffer index.

This function is only valid for buffers of type CARRAY, CSTRING, and XOCTET. Buffers of type FML and FML32 should use Do_TuxFMLData() to modify the contents of the buffer.

Syntax

```
int Do_Tuxsubstr ( int Index, char * Src, char * Dest );
```

Parameters

Parameter	Description
Index integer	Index into the buffer table.
Src char *	Pointer to string to search within indexed buffer.

Dest **char ***

Pointer to string to replace Src string if found.

Returns


int: Number of strings replaced in buffer index. 0 if string is not found in buffer index.

Example

```
Do_Tuxtpalloc( Buf1, "CARRAY", 1024);
Do_Tuxcarray( Buf1, "This is test data." );
printf( "%d, substitution made.\n", Do_Tuxsubstr ( Buf1, "test", "real" ) );
Do_TuxOutputBuffer( Buf1, 0, 20, 80 );
```

Output

```
"1 substitution made."
"[ 0]"This is real data.~~"
```

 **Note:** [xxxxx] precedes each output line to assist you in determining the current offset position in the buffer.

Do_Tuxtpabort

Aborts an active Tuxedo transaction.

Syntax

```
void Do_Tuxtpabort();
```

Parameters

None.

Example

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

Do_Tuxtpalloc

Calls a Tuxedo talloc command and stores the resultant address in a buffer table.

The `buffer_type` parameter is normally the standard Tuxedo type such as FML, FML32, etc. You can include a sub-type, if desired, by separating the type and the sub-type with a colon (:). The following is an example:

```
"MYTYPE:MYSUBTYPE".
```

Syntax

```
void Do_Tuxtpalloc( int buffer_index, char * buffer_type, int size );
```

Parameters

Parameter	Description
<code>buffer_index</code> int	Index into the buffer table.
<code>buffer_type</code> char *	Type of buffer: FML, FML32, CARRAY, etc.
<code>size</code> int	Number of bytes to allocate.

Example

```
Do_Tuxsetwsnaddr( "//LUCKY:3107" );
Do_Tuxtpinit( "Smith", "bapp", "passwd", "", TPU_DIP, "" );
Do_Tuxtpalloc( 1, "FML", 1024 );
Do_TuxFinit( 1 ); /* For: tpcall */
```

Do_Tuxtpbegin

Calls a Tuxedo tpbegin command.

Syntax

```
void Do_Tuxtpbegin( unsigned long timeout );
```

Parameters


Parameter	Description
timeout unsigned long	Number of seconds before the transaction times out.

Example

```
Do_Tuxtpbegin( 100 );
```

Do_Tuxtpbroadcast

Calls a Tuxedo tpbroadcast command.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpbroadcast( char * lmid, char * username, char cltname, int buffer_index, long flags );
```

Parameters


Parameter	Description
lmid char *	Logical machine ID, may be NULL to send the message to all users.
username char *	User name, may be NULL to send the message to all users.
cltname char *	Control group name, may be NULL to send the message to all control groups.
buffer_index integer	Index into the buffer table for the data to broadcast. Use buffer index 0 to indicate that no data should be sent with the broadcast.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_TuxFinit( 2 ); /* For: tpbroadcast */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtpbroadcast( "MYLMID", "MYUSER", "MYCLT", 2, TPNOTIME );
```


Do_Tuxtpcall

Calls a Tuxedo tpcall command. Prior to making the tpcall, it clears QALoad 's timer.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpcall( char * service, int in_buffer_idx, int out_buffer_idx, long flags );
```

Parameters

Parameter	Description
service char *	Tuxedo service name.
in_buffer_idx integer	Index into the buffer table for the input buffer.
out_buffer_idx integer	Index into the buffer table for the output buffer (may be the same as the input buffer).
flags integer	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_TuxFinit( 2 ); /* For: tpcall */
Do_TuxFMLData( test32_carray, 1, "~~~~~" );
Do_Tuxtpcall( "OPEN_TEST1", 2, 2, 0 );
```

Do_Tuxtppcommit

Calls a Tuxedo tppcommit command.

Syntax

```
void Do_Tuxtppcommit( );
```

Parameters

None.

Example


```
Do_Tuxtppbegin( 100 );
Do_Tuxtppcall( "OPEN_TEST1", 2, 2, 0 );
Do_Tuxtppcommit();
```

Do_Tuxtppconnect

Connects the client application to a conversational server.

The conn_index parameter is an index into a table of connection descriptors where the conversational descriptor is stored. This index is then used in subsequent conversational commands, such as Do_Tuxtppsend.

If you want to pass data to the server along with the connect, use a buffer allocated with DO_tppalloc. If you do not want to pass data to the server along with the connect, the buffer_index parameters should be specified as 0.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpconnect( int conn_index, char * service, int buffer_index, long flags );
```

Parameters

Parameter	Description
conn_index integer	Index into a table of connection descriptors. The first descriptor is 1.
service char *	Tuxedo service to connect to.
buffer_index integer	Index into the buffer table for data to be passed along with the connect. An index of zero means that no data passes.
flags integer	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_Tuxtpconnect( 1, "CON_TEST1", 0, TPSENDONLY );
Do_Tuxtpalloc( 2, "FML", 4096 );
Do_TuxFinit( 2 ); /* For: tpsend */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtpsend( 1, 2, TPRECVONLY );
```

Command Management

If you use conversational servers, you must include a connection descriptor in each tpsend and tprecv command. The descriptor returns when the connection is established by the tp_connect command. To assist you in maintaining these connection descriptors, the QALoad Tuxedo libraries store these descriptors in an internal connection table. Therefore, rather than specify a connection descriptor with each command, you simply specify the index into the table that contains the desired descriptor. The connection table, like the buffer table, has a lower bound of 1 and an upper bound that is dynamically allocated at runtime in the Do_TuxInitData command.

The following example illustrates how to use the connection table. Note that the example uses connection index 1 and Tuxedo buffer index 6.

```
/* Start a connection */
Do_Tuxtpconnect( 1, "CON_TEST1", 0, TPSENDONLY );
Do_Tuxtpalloc( 6, "FML", 4096 );

/*
 * Load FML data here.
 */

Do_Tuxtpsend( 1, 6, TPRECVONLY );
```

Each tpsend and tprecv command returns the integer value revent. This value indicates the status of the connection after the completion of the command. Use the Do_Tuxgetrevent command to retrieve this value.

Do_Tuxtpdiscon

Disconnects the client application from a conversational server.

The conn_index parameter points to an entry in the table of connection descriptors. The Do_Tuxtpconnect command loads the actual connection descriptor.

Syntax

```
void Do_Tuxtpdiscon( int conn_index );
```

Parameters


Parameter	Description
conn_index integer	Index into a table of connection descriptors. The first descriptor is 1.

Example

```
Do_Tuxtpconnect( 1, "CON_TEST1", 0, TPSENDONLY );
Do_Tuxtpalloc( 2, "FML", 4096 );
Do_TuxFinit( 2 ); /* For: tpsend */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtpsend( 1, 2, TPRECVOONLY );
Do_Tuxtpdiscon( 1 );
```

Do_Tuxtpenqueue

Enqueues a message to a Tuxedo queue. Prior to calling the Tuxedo command tpenqueue, Do_Tuxtpenqueue clears QALoad 's timer.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the ctl_flags and standard Tuxedo flags.

Syntax

```
void Do_Tuxtpenqueue( char * queuespace, char * queueName, long ctl_flags, long
ctl_priority, long ctl_urcode, char * ctl_replyqueue, char * ctl_failurequeue, int
buffer_index, long flags );
```

Parameters

Parameter	Description
queuespace char *	Queue space collection name.
queueName char *	Queue name.
ctl_flags long	Flags for the TPQCTL structure.
ctl_priority long	Enqueue priority in the TPQCTL structure.
ctl_urcode long	User return code in the TPQCTL structure.
ctl_replyqueue char *	Reply queue name in the TPQCTL structure.
ctl_failurequeue char *	Failure queue name in the TPQCTL structure.
buffer_index integer	Index into the buffer table. An index of zero means that no data passes.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)


Example

```
Do_TuxFinit( 2 ); /* For: tpenqueue */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtpenqueue( "QSPACE", "test1_queue", TPNOFLAGS, 2, 3, "replyq", "failq", 2, TPNOBLOCK
);
```

Do_Tuxtpinit

Calls a Tuxedo tpinit command.

The TPINFO buffer is allocated and released within the bounds of the Do_Tuxtpinit command, so there is no need to tpfree the buffer later in the script.

 **Note:** The atmi.h header file, usually located in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpinit ( char * username, char * ctlname, char password, char * grpname, int
flags, char * data );
```

Parameters


Parameter	Description
username char *	Tuxedo user name.
ctlname char *	Tuxedo client name.
password char *	Tuxedo password.
grpname char *	Tuxedo group name.
flags integer	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)
data char	An encoded char * that passes during the command. This is actual data, not a tpallocated buffer or buffer index.

Example

```
Do_Tuxsetwsnaddr( "//LUCKY:3107" );
Do_Tuxtpinit( "Smith", "bapp", "passwd", "", TPU_DIP, "" );
```

Do_Tuxtpost

Posts a Tuxedo event.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpost( char * event_name, int buffer_index, long flags );
```

Parameters

Parameter	Description
event_name char *	Tuxedo event name.
buffer_index integer	Index into the buffer table. An index of zero means that no data passes.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_TuxFinit( 2 ); /* For: tppost */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtppost( "ANYEVENTWITHDATA", 2, TPNOTRAN );
```

Do_Tuxtprealloc

Calls a Tuxedo tprealloc command to resize a previously allocated Tuxedo buffer.

Syntax

```
void Do_Tuxtprealloc( int buffer_index, int size );
```

Parameters

Parameter	Description
buffer_index integer	Index into the buffer table.
size integer	New buffer allocation size.

Example


```
Do_Tuxtprealloc( 1, 8192 );
```

Do_Tuxtprecv

Calls a Tuxedo tprecv command to receive data from a conversational server.

The connection descriptor table retrieves the connection descriptor and stores the returned data in a Tuxedo tallocated buffer pointed to by the buffer table.

Note that the Do_Tuxgetrevent command retrieves the global parameter revent.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtprecv( int conIndex, int index, long flags);
```

Parameters


Parameter	Description
conIndex integer	Index into a table of connection descriptors. The first descriptor is 1.
index integer	Index into the buffer table.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_Tuxtconnect( 1, "CON_TEST1", 0, TPSENDONLY );
Do_Tuxtpsend( 1, 2, TPRECVONLY ); /* Connection 1, buffer 2 */
Do_Tuxtprecv( 1, 2, 0 ); /* Connection 1, buffer 2 */
```

Do_Tuxtpscm

Calls the Tuxedo command tpscm. In turn, tpscm sets the TP_COMMIT_CONTROL characteristic to the value specified in flags.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpscm( long flags );
```

Parameters

Parameter	Description
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example


```
Do_Tuxtpscm( TP_CMT_LOGGED );
```

Do_Tuxtpsend

Calls a Tuxedo tpsend command which sends data to a conversational server.

The connection descriptors table retrieves the connection descriptor and stores the returned data in a Tuxedo tpallocated buffer pointed to by the buffer table.

Note that the DO_getrevent command retrieves the global parameter revent.

 **Note:** The atmi.h header file, usually found in the Tuxedo \include directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
void Do_Tuxtpsend ( int conn_index, int buffer_idx, long flags );
```

Parameters


Parameter	Description
conn_index integer	Index into a table of connection descriptors. The first descriptor is 1.
buffer_idx integer	Index into the buffer table.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_Tuxtppconnect( 1, "CON_TEST1", 0, TPSENDONLY );
Do_Tuxtppalloc( 2, "FML", 4096 );
Do_TuxFinit( 2 ); /* For: tpsend */
Do_TuxFMLData( test_string, 0, "Test string" );
Do_Tuxtppsend( 1, 2, TPRECVOONLY );
```

Do_Tuxtpprio

Calls the Tuxedo tpprio command which sets the priority for the next request sent.

 **Note:** The `atmi.h` header file, usually found in the Tuxedo `\include` directory, defines symbolic constants for the standard Tuxedo flags.

Syntax

```
Do_Tuxtsprio( int priority, long flags );
```

Parameters

Parameter	Description
priority integer	Priority for next request.
flags long	Tuxedo flags. (Please see Tuxedo documentation for valid flags.)

Example

```
Do_Tuxtsprio( 100, TPABSOLUTE );
```

Do_Tuxtpterm

Terminates QALoad 's connection to the Tuxedo server.

Syntax

```
void Do_Tuxtpterm();
```

Parameters

None.

Example

```
Do_Tuxtpterm();
```

Do_TuxUseCertificates

Used by the system to set the certificates flag when using Peoplesoft certificates. See `Do_Tuxcertsstr` for more information.

 **Note:** This command is generally used as internal to QALoad .

Syntax

```
void Do_TuxUseCertificates( int flag );
```

Parameters

flag **int**: 1 to use certificates, 0 to not use certificates.


Example

```
Do_TuxUseCertificates( 1 )
```

Do_Tuxxoctet

Loads the specified data into a `tpallocated` buffer of type `X_OCTET`.

If the buffer is too small to hold the data, `Do_Tuxxoctet` generates an error message.

 **Note:** Tuxedo buffer types `X_OCTET` and `Carray` are interchangeable when you create a script, the `Do_Tuxcarray` command will store data into buffers of type `X_OCTET`.

Syntax

```
void Do_Tuxxoctet( int buffer_idx, char * data );
```

Parameters

Parameter	Description
buffer_idx integer	Index into the buffer table.
data char *	Pointer to an encoded string containing the data to place into the buffer.

Example

```
Do_Tuxtppalloc( 3, "X_OCTET", 30 );
Do_Tuxxoctet( 3, "~&01&02&03"; )
```

Uniface

Uniface Index

BEGIN_UENTITY

Begins the declaration of the UNIFACE entity.

DO_Logfile_URB

Controls whether or not to generate a log file showing load test information, such as requests and responses.

DO_URB_AsciiToHex

Converts the ASCII hexadecimal represented buffer into its binary representation.

DO_URB_Init

Sets all necessary internal variables needed to load test a UNIFACE script.

DO_URB_setoprretry

Sets the number of times to retry an operation activation.

DO_URB_ubin2uf

Converts binary data to a UNIFACE format.

DO_URB_udbl2uf

Converts a double float to a UNIFACE format.

DO_URB_uecreate

Creates a UNIFACE environment.

DO_URB_uedelete

Closes a UNIFACE environment.

DO_URB_uentcreo

Creates an occurrence and makes it current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_uentoccs

Returns the number of occurrences that exist in an entity.

DO_URB_uentseto

Makes an occurrence current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_ufreeh

Deletes a handle.

DO_URB_uinstdel

Deletes a component instance.

DO_URB_uinstnew

Creates an instance of a component.

DO_URB_uinstopr

Returns a handle to an operation.

DO_URB_ulist2uf

Converts an item list to a UNIFACE format.

DO_URB_ulistdel

Deletes an item.

DO_URB_ulistfree

Frees a UNIFACE list.

DO_URB_ulistget

Gets an item.

DO_URB_ulistnew

Creates an item list.

DO_URB_ulistput

Puts an item.

DO_URB_ulistputlist

Copies an item from a specified source to the items of a list.

DO_URB_ulistputx

Puts an item.

DO_URB_ulong2uf

Converts a long to a UNIFACE format.

DO_URB_unifree

Frees memory.

DO_URB_uniname

Returns the name. Caller supplies allocated memory for name. Field names are not supported.

DO_URB_uopract

Activates an operation.

DO_URB_uoprprms

Returns the number of parameters.

DO_URB_uprmdir

Returns the direction of a parameter.

DO_URB_uprmgeth

Gets a reference to a parameter of an operation or a field of an entity. Or detaches a parameter from an operation.

DO_URB_uprmtype

Returns the data type of a parameter or entity field.

DO_URB_ustr2uf

Converts a string to a UNIFACE format.

DO_URB_uuf2bin

Converts a UNIFACE format to binary data.

DO_URB_uuf2dbl

Converts a UNIFACE format to a double float.

DO_URB_uuf2list

Converts a UNIFACE format to item list.

DO_URB_uuf2long

Converts a UNIFACE format to a long.

DO_URB_uuf2str

Converts a UNIFACE format to string.

END_UENTITY

Ends the declaration of a UNIFACE entity.

UFIELD

Declares or defines a UNIFACE field.

Uniface

BEGIN_UENTITY

Begins the declaration of the UNIFACE entity.

DO_Logfile_URB

Controls whether or not to generate a log file showing load test information, such as requests and responses.

DO_URB_AsciiToHex

Converts the ASCII hexadecimal represented buffer into its binary representation.

DO_URB_Init

Sets all necessary internal variables needed to load test a UNIFACE script.

DO_URB_setoprretry

Sets the number of times to retry an operation activation.

DO_URB_ubin2uf

Converts binary data to a UNIFACE format.

DO_URB_udbl2uf

Converts a double float to a UNIFACE format.

DO_URB_uecreate

Creates a UNIFACE environment.

DO_URB_uedelete

Closes a UNIFACE environment.

DO_URB_uentcreo

Creates an occurrence and makes it current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_uentoccs

Returns the number of occurrences that exist in an entity.

DO_URB_uentseto

Makes an occurrence current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_ufreeh

Deletes a handle.

DO_URB_uinstdel

Deletes a component instance.

DO_URB_uinstnew

Creates an instance of a component.

DO_URB_uinstopr

Returns a handle to an operation.

DO_URB_ulist2uf

Converts an item list to a UNIFACE format.

DO_URB_ulistdel

Deletes an item.

DO_URB_ulistfree

Frees a UNIFACE list.

DO_URB_ulistget

Gets an item.

DO_URB_ulistnew

Creates an item list.

DO_URB_ulistput

Puts an item.

DO_URB_ulistputlist

Copies an item from a specified source to the items of a list.

DO_URB_ulistputx

QALoad 5.02

Puts an item.

[DO_URB_ulong2uf](#)

Converts a long to a UNIFACE format.

[DO_URB_unifree](#)

Frees memory.

[DO_URB_uniname](#)

Returns the name. Caller supplies allocated memory for name. Field names are not supported.

[DO_URB_uopract](#)

Activates an operation.

[DO_URB_uoprprms](#)

Returns the number of parameters.

[DO_URB_uprmdir](#)

Returns the direction of a parameter.

[DO_URB_uprmgeth](#)

Gets a reference to a parameter of an operation or a field of an entity. Or detaches a parameter from an operation.

[DO_URB_uprmtype](#)

Returns the data type of a parameter or entity field.

[DO_URB_ustr2uf](#)

Converts a string to a UNIFACE format.

[DO_URB_uuf2bin](#)

Converts a UNIFACE format to binary data.

[DO_URB_uuf2dbl](#)

Converts a UNIFACE format to a double float.

[DO_URB_uuf2list](#)

Converts a UNIFACE format to item list.

[DO_URB_uuf2long](#)

Converts a UNIFACE format to a long.

[DO_URB_uuf2str](#)

Converts a UNIFACE format to string.

[END_UENTITY](#)

Ends the declaration of a UNIFACE entity.

[UFIELD](#)

Declares or defines a UNIFACE field.

[BEGIN_UENTITY](#)

Begins the declaration of a Uniface entity.

data	Pointer to a null terminated string containing the hexadecimal values of the bytes to be converted to binary.
------	---

Example

```
strcpy( urb_buffer, "0212035903107248");
DO_URB_AsciiToHex( urb_buffer );
DO_URB_ubin2uf( 4, 8, urb_buffer, 8 );
```

DO_URB_Init

Sets all necessary internal variables needed to load test a Uniface script.

Syntax

```
long DO_URB_Init(PPLAYERINFO *s_info);
```

Parameters

Parameter	Description
s_info	Pointer to a PLAYERINFO structure

Example

```
DO_URB_Init( s_info );
```

DO_URB_setoprretry

Sets the number of times to retry an operation activation.

Syntax

```
long DO_URB_setoprretry(int Retries, int Sleep);
```

Parameters

Parameter	Description
Retries	Number of times to retry operation.
Sleep	Time to sleep between retries.

DO_URB_ubin2uf

Converts binary data to a Uniface format.

Syntax

```
long DO_URB_ubin2uf(int nHandle, innt seqNr, char extData, long nLen);
```

Parameters

Parameter	Description
nHandle	Integer handle to an operation or entity.
seqNr	Integer sequence number.

extData	Char external binary data.
nLen	Long external data length.

Example

```
strcpy( urb_buffer, "0212035903107248");
DO_URB_AsciiToHex( urb_buffer );
DO_URB_ubin2uf( 4, 8, urb_buffer, 8 );
```

DO_URB_udbl2uf

Converts a double float to a Uniface format.

Syntax

```
DO_URB_udbl2uf( nHandle, seqNr, dData );
```

Parameters

Parameter	Description
nHandle	Integer handle to an operation or entity.
seqNr	Integer sequence number.
dData	External double float data.

Example

```
DO_URB_udbl2uf( 3, 1, 12345 );
```

DO_URB_uecreate

Creates a Uniface environment.

This function sets up a Uniface environment based on the configuration parameters: command line, assignment file, and working directory. Only one Uniface environment is allowed per process. The application start-up shell parameter apsName is ignored.

Syntax

```
DO_URB_uecreate( runmode, hInstance, cmdLine, asnName, apsName, workDir, envHandle )
```

Parameters

Parameter	Description
runMode	How Uniface is executed (batch or interactive). Always set this to 1.
hInstance	Instance handle. hInstance can be set to 0.
cmdLine	Command line.
asnName	.asn file name.
apsName	.aps file name.
workDir	Working directory.

envHandle	Uniface environment handle.
-----------	-----------------------------

Example

```
DO_URB_ucreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formula1\\formula1.asn", "", "c:\\u@training\\formula1\\formula1", 0 );
```

DO_URB_udelete

Closes a Uniface environment.

Syntax

```
long DO_URB_udelete(int envHandle,int level);
```

Parameters

Parameter	Description
envHandle	Uniface environment handle
level	Shutdown level.

Example

```
...
...
DO_URB_ucreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formula1\\formula1.asn", "", "c:\\u@training\\formula1\\formula1", 0 );
...
...
...
DO_URB_udelete( 0, -1 );
...
...
...

```

DO_URB_uentcreo

Creates an occurrence and makes it current. Note that this function can only be used with entity parameters and not with occurrence parameters.

Syntax

```
long DO_URB_uentcreo(int nHandle,long occNr);
```

Parameters

Parameter	Description
nHandle	Handle to the entity.
occNr	Sequence number of the occurrence.

Example

```
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "STORE ", 0, 2 ); |
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_uentcreo( 3, 1 ); / * creates first occurrence */
DO_URB_ustr2uf( 3, 1, "Field 1 data" );
```



```
DO_URB_ustr2uf( 3, 2, "Field 2 data" );
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
...
...
```

DO_URB_uentoccs

Gets the number of occurrences that exist in an entity.

Syntax

```
long DO_URB_uentoccs(int nHandle, long *occNr);
```

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to a Uniface entity.
occNr	(output) Number of occurrences that exist in the entity.

DO_URB_uentseto

Makes an occurrence current. Note that this function can only be used with entity parameters and not with occurrence parameters.

Syntax

```
long DO_URB_uentseto(int nHandle, long occNr);
```

Parameters

Parameter	Description
nHandle	Handle to an entity.
occNr	Sequence number of the occurrence.

Example

```
char *pString;
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_uentseto( 3, 1 ); /* make occurrence 1 current*/
DO_URB_uuf2str( 3, 1, pString );
...
... /* do some manipulation with the returned string */
...
DO_URB_unifree( 3, pString ); /* free the memory */
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_ufreeh

Deletes a handle.

Syntax

```
long DO_URB_ufreeh(int nHandle);
```

Parameters

Parameter	Description
nHandle	Any handle. Parameter hAny is always set to 0.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uinstdel

Deletes a component instance.

Syntax

```
long DO_URB_uinstdel(int nHandle);
```

Parameters

Parameter	Description
nHandle	Handle to a component instance.

Example

```
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
...
...
DO_URB_uinstdel( 1 );
DO_URB_ufreeh( 1 );
```

DO_URB_uinstnew

Creates an instance of a component.

The parameter options is UDEFAULT_COMM_MODE, USYNC_COMM_MODE, and so on. The parameter propList is a NULL-character separated item list. The item list ends in two NULL characters. The parameters compID, and propList are optional.

Syntax

```
DO_URB_uinstnew(envHandle, compName, compID, instName, options, propList, newHandle);
```

Parameters

Parameter	Description
-----------	-------------

envHandle	Uniface environment handle.
compName	Component name.
compID	Component ID.
instName	Instance name.
options	Options.
propList	Property list.
newHandle	Instance handle.

Example

```

...
...
DO_URB_ucreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formul1\\formul1.asn", "", "c:\\u@training\\formul1\\formul1", 0 );
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );

```

DO_URB_uinstopr

Returns a handle to an operation.

Syntax

```
long DO_URB_uinstopr(int nHandle,long oprName,int newHandle);
```

Parameters

Parameter	Description
nHandle	Handle to a component instance.
oprName	Operation name.
newHandle	Handle to the instance.

Example

```

...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
...
...
DO_URB_ufreeh( 2 );

```

DO_URB_uulist2uf

Converts an item list to a Uniface format.

Syntax

```
long DO_URB_uulist2uf(int nHandle,int SeqNr,int hList);
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
SeqNr	Sequence number.
hList	Handle to item list.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","", "",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uolistnew( 2, 3 ); /* FIELDS */
DO_URB_uolistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_uolistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_uolist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
```

DO_URB_uolistdel

Deletes an item.

This function corresponds with the Proc statement delitem. If UITEM_OPTION_NONE is specified, a value for index is expected and ID is ignored. If UITEM_OPTION_ID or UITEM_OPTION_ID_CASE is specified, index is ignored and a value for ID is expected.

Syntax

```
long DO_URB_uolistdel(int nHandle, int option, int index,char id);
```

Parameters

Parameter	Description
nHandle	Handle to an item list.
option	Option (UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE).
index	Item index.
id	pointer to an item ID.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","", "",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uopract( 2 );
...
...
DO_URB_uolistnew( 2, 3 ); /* FIELDS */
...
...
DO_URB_uuf2list( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_uolistdel( 3, UITEM_OPTION_NONE, 1, "");
```

```
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistfree

Frees a Uniface list.

Syntax

```
long DO_URB_ulistfree(int Handle);
```

Parameters

Parameter	Description
Handle	Handle to a list to be freed.

DO_URB_ulistget

Gets an item.

This function corresponds with the Proc statement `getitem`. The parameter option is `UITEM_OPTION_NONE`, `UITEM_OPTION_ID` or `UITEM_OPTION_ID_CASE`. If `UITEM_OPTION_NONE` is specified, a value for index is expected and ID is ignored. If `UITEM_OPTION_ID` or `UITEM_OPTION_ID_CASE` is specified, index is ignored and a value for ID is expected.

Syntax

```
DO_URB_ulistget(nHandle,option,index,id,pItem)
```

Parameters

Parameter	Description
nHandle	Handle to an item list.
option	Option (<code>UITEM_OPTION_NONE</code> , <code>UITEM_OPTION_ID</code> or <code>UITEM_OPTION_ID_CASE</code>).
index	Item index.
id	Pointer to an item ID.
pItem	Item

Example

```
char *pItem;
...
...
DO_URB_uinstnew(0,"S_SERVICE","", "",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uopract( 2 );
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
...
DO_URB_uuf2list( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ulistget( 3, UITEM_OPTION_NONE, 1, "", &pItem );
```

```

...
/* use pItem */
DO_URB_unifree( 3, pItem );
DO_URB_ufreeh( 3 );
...
...

```

DO_URB_ulistnew

Creates an item list.

Syntax

```
long DO_URB_ulistnew(int nHandle, int newHandle);
```

Parameters

Parameter	Description
nHandle	Any handle.
newHandle	A handle to an item list.

Example

```

DO_URB_uinstnew(0, "S_SERVICE", "", "", 1, "", 1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_ulist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
...
...

```

DO_URB_ulistput

Puts an item.

This function corresponds with the Proc statement putitem. The parameter option is UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE. If UITEM_OPTION_NONE is specified, a value for index is expected and ID is ignored. If UITEM_OPTION_ID or UITEM_OPTION_ID_CASE is specified, index is ignored and a value for ID is expected.

Syntax

```
long DO_URB_ulistput(int nHandle, int option, int index, char *id, char *item);
```

Parameters

Parameter	Description
nHandle	Handle to an item list.
option	Option (UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE).
index	Item index.

id	Pointer to an item ID.
item	Pointer to an item.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","", "",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_ulist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistputlist

Copies an item from a specified source to the items of a list.

Syntax

```
long DO_URB_ulistputlist(int nHandleDst, int index, char *id, int nHandleSrc);
```

Parameters

Parameter	Description
nHandleDst	Handle to the destination entity.
index	Item index.
id	Pointer to an item ID.
nHandleSrc	Handle to the source entity.

DO_URB_ulistputx

Puts an item.

Syntax

```
long DO_URB_ulistput(int nHandle, char *id, char *item, int sepcntr);
```

Parameters

Parameter	Description
nHandle	Handle to an item list.
id	Pointer to an item ID.
item	Pointer to an item.
sepcntr	Number of separators for list and sublists.

Example

```
DO_URB_uolistnew( 2, 3 ); /* FIELDS */
DO_URB_uolistputx( 3, "ORDER", "52", 2 );
```

DO_URB_ulong2uf

Converts a long to a Uniface format.

Syntax

```
long DO_URB_ulong2uf(int nHandle, int seqNr, long lData);
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
lData	External long data.

Example

```
DO_URB_ulong2uf(3,1,1234);
```

DO_URB_unifree

Frees memory.

Syntax

```
long DO_URB_unifree(int nHandle, void *pvoid);
```

Parameters

Parameter	Description
nHandle	Any handle.
pvoid	Pointer to start address of allocated memory.

Example

```
char *pString;
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_entseto( 3, 1 ); /* make occurrence 1 current*/
DO_URB_uuf2str( 3, 1, pString );
...
...
/* do some manipulation with the returned string */
...
DO_URB_unifree( 3, pString ); /* free the memory */
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```


DO_URB_uniname

Returns the name. Caller supplies allocated memory for name. Field names are not supported.

When working with an operation handle, if seqNr is 0, the name of the operation itself is returned. If seqNr is a legal parameter sequence number, the name of the parameter is returned.

Syntax

```
long DO_URB_uniname(int nHandle, int seqNr, int maxLen, char name);
```

Parameters

Parameter	Description
nHandle	Handle to a component instance, operation, or parameter.
seqNr	Sequence number.
maxLen	Size of name.
name	Name of the instance.

Example

```
char sName[128];
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uniname( 2, 1, 128, sName );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uopract

Activates an operation.

Syntax

```
long DO_URB_uopract(int nHandle);
```

Parameters

Parameter	Description
nHandle	Handle to an operation.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_ustr2uf( 2, 1, "19990101" );
DO_URB_ustr2uf( 2, 2, "19991231" );
DO_URB_uopract( 2 );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uoprprms

Returns the number of parameters.

Syntax

```
long DO_URB_uoprprms(int nHandle, long *pPrmCount);
```

Parameters

Parameter	Description
nHandle	Handle to an operation
pPrmCount	Pointer to number of parameters

Example

```
int nParameterCount;
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uoprprms( 2, &nParameterCount );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uprmdir

Returns the direction of a parameter.

Syntax

```
long DO_URB_uprmdir(int nHandle, int seqNr, int *pDirection );
```

Parameters

Parameter	Description
nHandle	Handle to an operation.
seqNr	Sequence number.
pDirection	Pointer to parameter direction. The parameter pDirection is UPARM_INPUT for IN direction, UPARM_OUTPUT for OUT direction, and (UPARM_INPUT UPARM_OUTPUT) for INOUT direction. It is not relevant whether or not parameter data is attached to the operation parameter list.

Example

```
int nDirection;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmdir( 3, &nDirection);
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_uprmgeth

Gets a reference to a parameter of an operation or a field of an entity. Or detaches a parameter from an operation.

Syntax

```
long DO_URB_uprmgeth(int nHandle, int seqNr, int bDetach, int newHandle );
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity parameter.
seqNr	Sequence number.
bDetach	Detach data option (TRUE/FALSE).
newHandle	Handle to a basic parameter, entity parameter or entity field. If parameter bDetach is TRUE, the data of handle nHandle is detached from the constructed handle nHandle. It is not allowed to detach a field from an entity.

Example

```
int nType;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmtype( 3, &nType);
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_uprmtype

Returns the data type of a parameter or entity field.

Syntax

```
long DO_URB_uprmtype(int nHandle, int *pType );
```

Parameters

Parameter	Description
nHandle	Handle to a parameter or entity field.
pType	Pointer to data type. The parameter pType is UTYPE_STRING, UTYPE_BOOLEAN, and so on.

Example

```
int nType;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmtype( 3, &nType);
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

QALoad 5.02

```
DO_URB_ufreeh( 3 );  
...
```

DO_URB_ustr2uf

Converts a string to a Uniface format.

Syntax

```
long DO_URB_ustr2uf(int nHandle,int seqNr,char *string)
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
string	Pointer to an external string data.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);  
DO_URB_uinstopr( 1, "LIST", 0, 2 );  
DO_URB_ustr2uf( 2, 1, "19990101" );
```

DO_URB_uuf2bin

Converts a Uniface format to binary data.

Syntax

```
long DO_URB_uuf2bin(int nHandle, int seqNr, char pExData, long *pnLen);
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
pExData	External (output) binary data. The parameter pExtData is allocated on the heap. It has to be freed with DO_URB_uifree.
pnLen	Pointer to external data length.

Example

```
char *pBinaryData;  
long nLen;  
...  
...  
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);  
DO_URB_uinstopr( 1, "RETRIEVE", 0, 2 );  
DO_URB_uopract( 2 );  
DO_URB_uuf2bin ( 2, 4, &pBinaryData, &nLen );  
...  
...
```

```
DO_URB_unifree( 2 , pBinaryData );
DO_URB_ufreeh( 2 );
```

DO_URB_uuf2dbl

Converts a Uniface format to a double float.

Syntax

```
long DO_URB_uuf2dbl(int nHandle,int seqNr,double *pdData);
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
pdData	Pointer to an external double float data.

Example

```
double dNumber;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2dbl ( 2, 2, &dNumber );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uuf2list

Converts a Uniface format to item list.

Syntax

```
long DO_URB_uuf2list(int nHandle, int seqNr, int hList);
```

Parameters

Parameter	Description
nHandle	Handle to an operation or entity
seqNr	Sequence number
hList	Create the parameter hList with DO_URB_ulistnew before calling this function. After using the item list, free it using DO_URB_ulistdel

Example

```
char *pItem;
...
...
DO_URB_uinstnew(0,"S_SERVICE","", "",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...

```


pExString	External string data. The parameter pExString is allocated on the heap. It has to be freed with DO_URB_unifree.
-----------	---

Example

```
char *pString;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2str ( 2, 3, &pString );
..
..
DO_URB_unifree( 2 , pString );
DO_URB_ufreeh( 2 );
```

END_UENTITY

Ends the declaration of a Uniface entity.

Syntax

```
END_UENTITY(UNIFACE_ENTITY* pointer, UNIFACE_ENTITY* pointee);
```

Parameters

Parameter	Description
pointer	New pointer for the entity.
pointee	The original pointer used in BEGIN_UENTITY as entname.

Example

```
END_UENTITY(RACE_FORMULA1, race_formula1);
```

UFIELD

Declares or defines a Uniface field.

Syntax

```
UFIELD (char* name, char* type, int size, char* index);
```

Parameters


Parameter	Description
name	Field name.
type	Data and interface type.
size	Length of the field.
index	If the field is a key for the entity, this field holds the index.

Example

```
UFIELD( "RACE_ID", "N2", 4, "1.101" );
```

Uniface Polyserver (Versions 7.2.04 - 7.2.06)

Uniface Polyserver Index

 **Note:** These commands apply only to Uniface Versions 7.2.04 through 7.2.06.

[DO_Logfile_PSV](#)

Starts or stops the call logging.

[DO_PSV_bin2uf](#)

Converts binary data to a Uniface field.

[DO_PSV_clean](#)

Cleans the Polyserver middleware interface.

[DO_PSV_close](#)

Close at least one database table.

[DO_PSV_commit](#)

Commits a transaction.

[DO_PSV_creocc](#)

Creates an empty occurrence in the hitlist.

[DO_PSV_delete](#)

Deletes a record.

[DO_PSV_fclose](#)

Closes a file on a server.

[DO_PSV_fetch](#)

Fetches a record from a table or locks a record.

[DO_PSV_ffield](#)

Deletes a file from the server.

[DO_PSV_fopen](#)

Opens a file on the server.

[DO_PSV_fread](#)

Reads a file.

[DO_PSV_free](#)

Frees all allocated memory for a specified entity.

[DO_PSV_fwrite](#)

Writes a stream of bytes to a polyserver file.

[DO_PSV_init](#)

Initializes the Polyserver middleware interface.

[DO_PSV_logoff](#)

Closes a polyserver path previously opened with a call to DO_PSV_logon.

DO_PSV_logon

Opens a path to a polyserver.

DO_PSV_long2uf

Converts a long number to a Uniface field.

DO_PSV_modify

Modifies the mode of action of a database table.

DO_PSV_ndelete

Deletes or nullifies a set of records.

DO_PSV_open

Opens or creates a table, or generates the Data Definition Language (DDL) to create a table or creates SQL scripts to handle referential constraints.

DO_PSV_remotepath

Provides and prepares an encrypted path for a remote assignment.

DO_PSV_rollback

Rolls back the transaction.

DO_PSV_select

Selects records from a table that match a specified profile.

DO_PSV_selectdb

Select using aggregate functions.

DO_PSV_setocc

Makes the specified occurrence current.

DO_PSV_sql

Submits a DML statement to the DBMS.

DO_PSV_sselect

Performs a single select on the specified entity.

DO_PSV_str2uf

Converts a C string into a Uniface field.

DO_PSV_uf2bin

Converts a Uniface field into its binary representation.

DO_PSV_uf2dbl

Converts a Uniface field into a double.

DO_PSV_uf2long

Converts a Uniface field into a long.

DO_PSV_uf2str

Converts a Uniface field into a C null terminated string.

DO_PSV_update

Updates the current record.

DO_PSV_write

Inserts a new record.

DO_PSV_xtrans

Sends an X transaction to a polyserver.

DO_PSV_zero

Sends a 0 transaction to a polyserver.

DO_Logfile_PSV

Starts or stops the call logging.

DO_Logfile_PSV starts or stops the logging of the Polyserver middleware interface calls. The calls are logged in a file named psvnnnn.cap, where n is the virtual user number.

Syntax

```
int DO_Logfile_PSV(int flag);
```

Parameters

Parameter	Description
flag	TRUE or FALSE

Example

```
DO_Logfile_PSV(TRUE);
```

DO_PSV_bin2uf

Converts binary data to a Uniface field.

DO_PSV_bin2uf converts binary data into a Uniface field. If the name of the field is not a field of the entity, it is automatically associated as a local variable for the entity and replaced inside where clauses when the name appears between "%%" and "%%%".

Syntax

```
int DO_PSV_bin2uf(UNIFACE_ENTITY *pentity, char *fieldname, char *value, int size);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.

- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
entity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field or local variable that will get the data.
value	The value to be converted to a Uniface field.
size	The size of the data buffer to convert.

Example

```
char *pBin1
pBin = "1234567890";
DO_PSV_bin2uf(ONE_ENTITY_PKTEST_SZ1, "F1", pBin, 10);
```

DO_PSV_clean

Cleans the Polyserver middleware interface.

DO_PSV_clean clears and unloads the network layer from the interface.

Syntax

```
int DO_PSV_clean(POLY_PROTOCOL *pproto);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
-----------	-------------

pproto	A pointer to a protocol structure. pproto has to be initialized with the DO_PSV_init call.
--------	--

Example

```
int rrobot_script(s_info);
PLAYER_INFO *s_info;

{
...
POLY_PROTOCOL PROTO;
...
DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

{script body ...}

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}
```

DO_PSV_close

Close at least one database table.

DO_PSV_close requests the polyserver process to close all the opened tables passed as parameters to the call. The "..." as a parameter means that the number of parameters for this call is not fixed.

The parameter list has to be terminated by the NULL parameter. If the NULL parameter does not terminate the list, the API keeps reading the stack for extra parameters, causing unexpected results.

Syntax

```
int DO_PSV_close(UNIFACE_ENTITY *pentity, char *mode, UNIFACE_ENTITY *pentity, ...);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	Pointer to a base UNIFACE_ENTITY structure to be closed.
mode	The mode to be used to close the table(s). For more explanation on the mode parameter, refer to the Uniface Database Driver Cookbook.
pexterity	A pointer to a UNIFACE_ENTITY to be closed.
(...)	pexterity pointer(s) to extra UNIFACE_ENTITY to be closed.

Example

```

int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_si1, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_si1);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1","ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.1 01,F1,S2, 0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

BEGIN_UENTITY(many_ent_pktest_sz1, "MANY_ENT", "PKTEST",
"SZ1","MANY_ENT,PKTEST,SZ1,PK,N2,M,10,100,2, 1.101,FK,N2,M,10,100,2,1.102,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("FK", "N2", 2, "1.102")
UFIELD("F1", "S2", 20, "")
END_UENTITY(MANY_ENT_PKTEST_SZ1, many_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);
BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );
...
DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
...
DO_PSV_open(&PATH_1, MANY_ENT_PKTEST_SZ1, "00C8" );
{script body ...}
...
DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, MANY_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();
DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_commit

Commits a transaction.

Commits the transaction on the logon path specified by the pentity and pentity. Note that the last entity specified in the parameter list has to be the NULL value. Not inserting the NULL value as the last parameter produces unexpected results.

Syntax

```
int DO_PSV_commit(UNIFACE_ENTITY *pentity, char *mode, UNIFACE_ENTITY *pentity, ..., NULL);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A base pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution. 0000: perform a normal commit. 0001: perform the first phase of a two phase commit. 0002: perform the second phase of a two phase commit.
pentity	Pointer(s) to UNIFACE_ENTITY structure(s) describing extra tables participating in the commit.

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;
BEGIN_ENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_ENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);
BEGIN_ENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1", "ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
```

```

UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_setocc(ONE_ENT_PKTEST, -1);
DO_PSV_fetch(ONE_ENT_PKTEST, "0004", "0022");
DO_PSV_setocc(ONE_ENT_PKTEST, -1);
DO_PSV_fetch(ONE_ENT_PKTEST, "0004", "0022");
DO_PSV_str2uf(MANY_ENT_PKTEST, "FK", "9");
DO_PSV_ndelete(MANY_ENT_PKTEST, "0000", "");
DO_PSV_delete(ONE_ENT_PKTEST, "0000", "0422");

{script body ...}

...

DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();
DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_creocc

Creates an empty occurrence in the hitlist.

DO_PSV_creocc creates an empty occurrence in the hitlist of the entity identified by pentity.

Syntax

```
int DO_PSV_creocc(UNIFACE_ENTITY *pentity);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.

- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.

Example

```
DO_PSV_creocc(MANY_ENT_PKTEST_SZ1);
```

DO_PSV_delete

Deletes a record.

DO_PSV_delete deletes the current record in the table specified by pentity.

Syntax

```
int DO_PSV_delete(UNIFACE_ENTITY *pentity, char *mode);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution (0000 always zero).

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...

```



```

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1", "ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_setocc(ONE_ENT_PKTEST, -1);
DO_PSV_fetch(ONE_ENT_PKTEST, "0004", "0022");
DO_PSV_delete(ONE_ENT_PKTEST, "0000", "0422");

{script body ...}

...

DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_fclose

Closes a file on a server.

Syntax

```
int DO_PSV_fclose(PLOGON_PATH pnet,int fp);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.

- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
fp	A handle to a file previously opened using the DO_PSV_fopen call.

Example

```
fp = DO_PSV_fopen(&PATH_1 , "w_start.frm", "0000F");
DO_PSV_fread(&PATH_1, fp, "0000", "1000", NULL, 0);
DO_PSV_fread(&PATH_1, fp, "0000", "0200", NULL, 0);
DO_PSV_fclose(&PATH_1, fp);
```

DO_PSV_fetch

Fetches a record from a table or locks a record.

The current record in the hit list identifies exactly one record. This is the record to be fetched.

Syntax

```
int DO_PSV_fetch(UNIFACE_ENTITY *pentity, char *mode, char *hitstatus);
```

Return Value

0 = successful
 < 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure that has been previously

	initialized by a call to DO_PSV_open.
mode	The mode of execution. 0000: fetches the current record in the hitlist. Update the entry in the hitlist. 0001: fetches the current record and locks it exclusively. 0002: locks the current record without fetching the data.

DO_PSV_fkill

Deletes a file from the server.

Syntax

```
int DO_PSV_fkill(PLOGON_PATH pnet, char *name, char *mode);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
name	The name of the file on the server.
mode	The Uniface mode used to delete the file. For more description on the mode, refer to the Uniface Database Driver Cookbook.

Example

```
DO_PSV_fkill(&PATH_1, "w_start.frm", "00000");
```

DO_PSV_fopen

Opens a file on the server.

DO_PSV_open will open a file on the server. The call returns a file handle to be used in the subsequent file related calls.

Syntax

```
int DO_PSV_fopen(PLOGON_PATH pnet, char *name,char *mode);
```

Return Value

0 = failure
> 0 = handle to the file

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
name	The name of the file on the server.
mode	The Uniface mode used to delete the file. For more description on the mode, refer to the Uniface Database Driver Cookbook.

Example

```
fp = DO_PSV_fopen(&PATH_1 , "w_start.frm", "0000F");
DO_PSV_fread(&PATH_1, fp, "0000", "1000", NULL, 0);
DO_PSV_fread(&PATH_1, fp, "0000", "0200", NULL, 0);
DO_PSV_fclose(&PATH_1, fp);
```

DO_PSV_fread

See also [Uniface Polyserver](#)

Reads a file.

DO_PSV_fread reads a specified number of bytes from a file on the server.

Syntax

```
int DO_PSV_fread(PLOGON_PATH pnet, int fp,char *type, char *mode,char *buffer,int size);
```

Return Value

0 = number of bytes read
< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.

- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
fp	A handle to a file previously opened using the DO_PSV_fopen call.
type	Type read.
mode	Mode read.
buffer	A pointer to a buffer that will receive the content of the file.
size	The size of the buffer

Example

```
fp = DO_PSV_fopen(&PATH_1 , "w_start.frm", "0000F");
DO_PSV_fread(&PATH_1, fp, "0000", "1000", NULL, 0);
DO_PSV_fread(&PATH_1, fp, "0000", "0200", NULL, 0);
DO_PSV_fclose(&PATH_1, fp);
```

DO_PSV_free

Frees all allocated memory for a specified entity.

Syntax

```
int DO_PSV_free(UNIFACE_ENTITY *pentity);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.

Example

```
DO_PSV_free($DEF_UNIFACE_SI1);
```

DO_PSV_fwrite

Writes a stream of bytes to a polyserver file.

Syntax

```
DO_PSV_fwrite(PLOGON_PATH pnte, int fp, char *type, char mode, char *buffer, int size);
```

Return Value

0 = number of bytes read

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
fp	A handle to a file previously opened using the DO_PSV_fopen call.
type	Type read.
mode	Mode read.
buffer	A pointer to a buffer that will receive the content of the file.
size	The size of the buffer.

DO_PSV_init

Initializes the Polyserver middleware interface.

DO_PSV_init initializes the middleware interface. Only POLY_NET_TCP network interface is currently supported.

The IOLevel is used to ask the Polyserver to return the IO debugging messages back to the client interface. The messages are displayed on the Player window when the Debug Trace option is selected on the Conductor's Test Information Window, Script Assignment tab. Use the IOLevel setting for debugging purposes only. Having an IOLevel greater than zero may affect the timing result during a load test.

Syntax

```
int DO_PSV_init(POLY_PROTOCOL *pproto, int type, PPLAYER_INFO s_info, int Iolevel);
```

Return Value

0 = successful

< = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pproto	Pointer to a POLY_PROTOCOL structure.
int type	Interface type, only POLY_NET_TCP is currently supported.
s_info	Player info structure.
int Iolevel	Polyserver debugging IO Level (0-512).

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
...
DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);
```

QALoad 5.02

```
{script body ...}  
DO_PSV_clean(&PROTO);  
REPORT(SUCCESS);  
EXIT();  
return(0);  
}
```

DO_PSV_logoff

Closes a polyserver path previously opened with a call to DO_PSV_logon.

Syntax

```
int DO_PSV_logoff(PLOGON_PATH pnet);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure previously initialized with a call to DO_PSV_logon.

Example

```
int rrobot_script(s_info)  
PLAYER_INFO *s_info;  
{  
...  
POLY_PROTOCOL PROTO;  
LOGON_PATH PATH_1;  
...  
DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);  
BEGIN_TRANSACTION();  
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
```



```
{script body ...}
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}
```

DO_PSV_logon

Creates and opens a new path to a polyserver process.

A Uniface application may open multiple paths to the same host. If a Uniface entity is defined on a path that has not yet been opened, a DO_PSV_logon call is needed. If the Uniface application requires multiple paths to be opened for multiple transaction handling, a DO_PSV_logon call is needed as well.

Syntax

```
int DO_PSV_logon(POLY_PROTOCOL *pproto, PLOGON_PATH pnet, char *host, char *user, char *passwd, char *path);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pproto	A pointer to a POLY_PROTOCOL structure previously initialized with a call to DO_PSV_init.
pnet	A pointer to a LOGON_PATH structure that contains all the information related to the opened path.
host	The name of the host server running the polyserver process. The form hostname+port is accepted. If the port is not specified, the interface searches the current client services file for the polyserver entry.

user	User ID.
passwd	Password.
path	The name of the path the server logon process uses to create the polyserver process on the host.

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;

{
...

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

...

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);
BEGIN_TRANSACTION();

DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
{script body ...}
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}
```

DO_PSV_long2uf

Converts a long number to a Uniface field.

DO_PSV_long2uf converts a number with a long representation to a Uniface field belonging to the entity pentity. If the name of the field is not a field of the entity, it is automatically associated as a local variable for the entity and replaced inside where clauses when the name appears between "%%" and "%%%".

Syntax

```
int DO_PSV_long2uf(UNIFACE_ENTITY *pentity, char *fieldname, long lValue);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.

- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field or local variable that gets the value.
lValue	The value.

Example

```
long lvalue;
DO_PSV_long2uf( ONE_ENTITY_PKTEST_SZ1, "PK", lvalue);
```

DO_PSV_modify

Modifies the mode of action of a database table.

DO_PSV_modify modifies the way Uniface manipulates the data of a database entity. It is strongly recommended not to change this call, since it may change the behavior of the polyserver, thus not reproducing accurate performance results.

Note that the DO_PSV_modify call does not expect any return values from the host, making timing values for this call irrelevant.

The modify calls are generated when a component has different entity properties.

Syntax

```
int DO_PSV_modify(UNIFACE_ENTITY *pentity, char *extrainfo);
```

Return Value

None

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure that has previously been initialized with a call to DO_PSV_open.
extrainfo	Binary information to be sent to the polyserver process: Static 4 total IDs total IDs *4 IDs 1 noupdate 1 readbykey

	1	udynopt
	1	release
	4	listfld
	listfld *1	fieldsubsetting (1/0)

Example

```

int rhobot_script(s_info)
PLAYER_INFO *s_info;

{
...

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1","ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

BEGIN_UENTITY(many_ent_pktest_sz1, "MANY_ENT", "PKTEST",
"SZ1","MANY_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,FK, N2,M,10,100,2,1.102,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("FK", "N2", 2, "1.102")
UFIELD("F1", "S2", 20, "")
END_UENTITY(MANY_ENT_PKTEST_SZ1, many_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();

DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_open(&PATH_1, MANY_ENT_PKTEST_SZ1, "00C8" );

{script body ...}

...

DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1, ONE_ENT_PKTEST_SZ1,
MANY_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);

```

DO_PSV_ndelete

Deletes or nullifies a set of records.

DO_PSV_ndelete deletes or nullifies a set of records defined by the section profile. A profile is built using the current occurrence of the table represented by pentity and the where clause.

Syntax

```
int DO_PSV_ndelete(UNIFACE_ENTITY *pentity, char *mode, char *where);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution (0000 always zero).
where	The where clause associated with the select. If it starts with WHERE, then the dml statement is sent to the database. If it starts with 'U_WHERE Uniface takes care of generating the appropriate select request for the database.

DO_PSV_open

Opens or creates a table, or generates the Data Definition Language (DDL) to create a table or creates SL scripts to handle referential constraints.

DO_PSV_open creates and initializes a UNIFACE_ENTITY structure that holds the information that relates to a particular entity in the Uniface application. The entity structure holds name, model, and a hit list for the entity.

Syntax

```
int DO_PSV_open(PLOGON_PATH pnet, UNIFACE_ENTITY *pentity, char *mode);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH previously opened with a call to DO_PSV_logon.
pentity	A pointer to a UNIFACE_ENTITY. pentity has to be created and defined using the macros BEGIN_UENTITY, UFIELD, and END_UENTITY. Failing to use the macros to create the structure generates unexpected results.
mode	The Uniface mode used to open the database table. For more description on the mode, refer to the Uniface Database Driver Cookbook.

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;

{
...

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1", "ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.1 01,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

BEGIN_UENTITY(many_ent_pktest_sz1, "MANY_ENT", "PKTEST",
"SZ1", "MANY_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1. 101,FK,N2,M,10,100,2,1.102,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("FK", "N2", 2, "1.102")
UFIELD("F1", "S2", 20, "")
END_UENTITY(MANY_ENT_PKTEST_SZ1, many_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
```

```

DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );
...
DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
...
DO_PSV_open(&PATH_1, MANY_ENT_PKTEST_SZ1, "00C8" );
{script body ...}
...
DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, MANY_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_remotepath

Provides and prepares an encrypted path for a remote assignment.

Syntax

```
int DO_URB_remotepath(PLOGON_PATH pnet, char *encrpath);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pnet	A pointer to a LOGON_PATH structure representing an opened path to the server.
encrpath	The encrypted path.

DO_PSV_rollback

Rolls back the transaction.

Rolls back the transaction on the logon path specified by pentity and pentity. Note that the last entity specified in the parameter list has to be the NULL value. Not inserting the NULL value as the last parameter produces unexpected results.

Syntax

```
int DO_PSV_rollback(UNIFACE_ENTITY *pentity, char *mode, UNIFACE_ENTITY *pentity,
...,NULL);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A base pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution (0000 always zero).
pentity	Pointer(s) to UNIFACE_ENTITY structure(s) describing extra tables participating in the rollback.

Example

```
DO_PSV_rollback(ONE_ENT_PKTEST, "0000", ONE_ENT_PKTEST, MANY_ENT_PKTEST, SYSPK_PKTEST,
NULL);
```

DO_PSV_select

Selects records from a table that match a specified profile.

DO_PSV_select sends a select request for the specified database table. The select request builds a search profile based on the values stored in the current occurrence of the specified entity. Depending on the mode of action, the select request adds new hits to the hit list for the entity, or discards the hit list prior to making the call.

The notion of local variable is also available. This allows the programmer to move values into variables associated with the entity and have the DO_PSV_select call to replace them in the where clause. The replacement is defined as a Uniface string insertion delimited by "%%" and "%%%".

Any name but actual field names of the current entity or valid tokens can be used to define local variables. The convert process automatically generates a local variable when it recognizes a static value in a where clause. The variables name starts with the "\$" sign followed by a number.

Syntax

```
int DO_PSV_select(UNIFACE_ENTITY *pentity, char *mode, char *maxhits, char *cachesize, char *where, char *orderby, char *option);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY that has been previously initialized by a call to DO_PSV_open.
mode	The mode of execution: 0000: select the first step of the hitlist. 0001: identical to mode 0000, but do not add records to the hitlist. 0002: profile match 0003: select the next step in the hitlist.
maxhits	The number of record to select in this step if 0, select all matching records.
cachesize	The size of the cache to hold a hit.
where	The where clause associated with the select. If it starts with WHERE, then the dml statement is sent to the database. If it starts with U_WHERE, Uniface taks care of generating the appropriate select request for the database.
orderby	The order by clause of the select statement.

option

The option clause associated with the select statement.

Example

```

int rrobot_script(s_info)
PLAYER_INFO *s_info;

{
...

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1",
"$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1","ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

BEGIN_UENTITY(many_ent_pktest_sz1, "MANY_ENT", "PKTEST",
"SZ1","MANY_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,FK, N2,M,10,100,2,1.102,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("FK", "N2", 2, "1.102")
UFIELD("F1", "S2", 20, "")
END_UENTITY(MANY_ENT_PKTEST_SZ1, many_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_open(&PATH_1, MANY_ENT_PKTEST_SZ1, "00C8" );

/* This select sample code creates an empty occurrence of an
entity called MANY_ENT_PKTEST, a retrieve profile is
created by setting the value of a field using
DO_PSV_str2uf. As well, a U_WHERE and ORDER BY clauses are
added to the select. When using an SQL compliant database,
the generated SQL statement for this example may look
like:

SELECT PK, FK, F1
FROM MANY_ENT_PKTEST
WHERE (FK = 1 AND F1 != 'HELLO')
ORDER BY F1 DESC

DO_PSV_creocc(MANY_ENT_PKTEST_SZ1);
DO_PSV_str2uf(MANY_ENT_PKTEST_SZ1, "FK","1");

/* read U_WHERE( F1 != "HELLO")
*
* options
*/

DO_PSV_select(MANY_ENT_PKTEST_SZ1, "0000", "000A", "0200", "U_WHERE( F1 != \"HELLO\")",
"ORDER BY PK DESC", "");

{script body ...}

...

```

```
DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1, ONE_ENT_PKTEST_SZ1,
MANY_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}
```

DO_PSV_selectdb

Select using aggregate functions.

DO_PSV_selectdb sends aggregate requests to the server. Supported functions are:

- ! TRANSPORT — Passes the value of the specified field of the last selected record.
- ! SUM — Computes the sum of the specified field in the selected records.
- ! MAX — Computes the maximum value of the specified field in the selected records.
- ! MIN — Computes the minimum value of the specified field in the selected records.
- ! COUNT — Computes the number of selected records that contain a value in the specified field.
- ! AVG — Computes the average value of the specified field in the selected records.

Syntax

```
int DO_PSV_selectdb(UNIFACE_ENTITY *pentity, char *mode, char *aggregate, char *where, char
*option);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY previously initialized with a call to DO_PSV_open and associated with the selectdb call.

aggregate	A string representing the aggregate selection.
where	A where clause associated with the selection of records. See DO_PSV_select for more information on the where clause. Local variables are accepted.

Example

```

int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1","ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1, S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_selectdb(ONE_ENT_PKTEST_SZ1,"0000", "MAX(PK)", "", "");

{script body ...}

...

DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_setocc

Makes the specified occurrence current.

DO_PSV_setocc makes the occurrence specified by occ current in the pentity entity. If the occ is greater than the last possible occurrence, the last occurrence is made current.

Syntax

```
int DO_PSV_setocc(UNIFACE_ENTITY *pentity, int occ);
```

Return Value

0 = success occurrence number

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure,
occ	An occurrence number. -1 means the last occurrence.

Example

```
DO_PSV_setocc(SYSPK_PKTEST, 1);
```

DO_PSV_sql

Submits a DML statement to the DBMS.

DO_PSV_sql sends a DML statement to the DBMS on the opened path. The replacement of local variables is allowed with this call.

Syntax

```
int DO_PSV_sql(UNIFACE_ENTITY *pentity, char *mode, char *statement, char *path);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.

- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution: 0000: sends a statement in the native Data Manipulation Language (DML) to the DBMS. 0001: reserved. 0002: sends a statement in the native DML to the DBMS. If the statement selects data from the database, the column name will appear in the returned rows. 0003: gets subsequent set of rows.
statement	A valid SQL statement.
path	A valid Uniface path.

Example

```
LOGON_PATH PATH_1;
BEGIN_ENTITY($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_ENTITY($DEF_UNIFACE_SI1, $def_uniface_sil);
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

/* sql "select count(*) from one_ent", "DEF" */
DO_PSV_sql($DEF_UNIFACE_SI1, "0000", "select count(*) from one_ent", "DEF");
```

DO_PSV_sselect

Performs a single select on the specified entity.

See the description of [DO_PSV_select](#) for a definition of the single select call.

Syntax

```
int DO_PSV_sselect(UNIFACE_ENTITY *pentity, char *mode, char *where, char *orderby, char *option);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.

- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
penity	A pointer to a UNIFACE_ENTITY that has been previously initialized by a call to DO_PSV_open.
mode	The mode of execution: 0000 select the first step of the hitlist 0001 identical to mode 0000 but do not add records to the hitlist 0002 profile match 0003 select the next step in the hitlist.
where	The where clause associated with the select. If it starts by 'WHERE' that the dml statement is sent to the database, if it starts by 'U_WHERE' Uniface take care of generating the appropriate select request for the database.
orderby	The order by clause of the select statement.
option	The option clause associated with the select statement.

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UEntity($def_uniface_sil, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UEntity($DEF_UNIFACE_SI1, $def_uniface_sil);

BEGIN_UEntity(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1","ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UEntity(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

BEGIN_UEntity(many_ent_pktest_sz1, "MANY_ENT", "PKTEST",
"SZ1","MANY_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,FK, N2,M,10,100,2,1.102,F1,S2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("FK", "N2", 2, "1.102")
UFIELD("F1", "S2", 20, "")
END_UEntity(MANY_ENT_PKTEST_SZ1, many_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_open(&PATH_1, MANY_ENT_PKTEST_SZ1, "00C8" );
```

QALoad 5.02

/* This select sample code creates an empty occurrence of an entity called MANY_ENT_PKTEST, a retrieve profile is created by setting the value of a field using DO_PSV_str2uf. As well, U_WHERE and ORDER BY clauses are added to the select. When using an SQL compliant database, the generated SQL statement for this example may look like:

```
SELECT PK, FK, F1
FROM MANY_ENT_PKTEST
WHERE (FK = 1 AND F1 != 'HELLO')
ORDER BY F1 DESC

DO_PSV_creocc(MANY_ENT_PKTEST_SZ1);
DO_PSV_str2uf(MANY_ENT_PKTEST_SZ1, "FK", "1");

/* read U_WHERE( F1 != "HELLO" )
*
* options
*/

DO_PSV_select(MANY_ENT_PKTEST_SZ1, "0000", "000A", "0200", "U_WHERE( F1 != \"HELLO\")",
"ORDER BY PK DESC", "");

{script body ...}

...

DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1, ONE_ENT_PKTEST_SZ1,
MANY_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}
```

DO_PSV_str2uf

Converts a C string into a Uniface field.

DO_PSV_str2uf converts a null terminated string into a Uniface field. If the destination field in the entity pentity is a date, datetime, or time field, it is converted to the proper internal Uniface date/time format. If the name of the field is not a field of the entity, it is automatically associated as a local variable for the entity and replaced inside where clauses when the name appears between "%%" and "%%%".

Syntax

```
int DO_PSV_str2uf(UNIFACE_ENTITY *pentity, char *fieldname, char *string);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.

- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field or local variable that gets the string.
string	The string to be converted to a Uniface field.

Example

```
DO_PSV_str2uf(MANY_ENT_PKTEST_SZ1, "FK", "1");
```

DO_PSV_uf2bin

Converts a Uniface field into its binary representation.

Syntax

```
int DO_PSV_uf2bin(UNIFACE_ENTITY *pentity, char *fieldname, char *dst, int dsize);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
-----------	-------------

pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field.
dst	A pointer to a buffer that receives the data.
dsize	The size of the buffer that receives the data.

Example

```
#define BIN_AREA_SIZE 2048
char bin_data[BIN_AREA_SIZE];
DO_PSV_uf2bin(TRACK_FORMULA1_SZ1, "TRACK_MAP", bin_data, BIN_AREA_SIZE);
```

DO_PSV_uf2dbl

Converts a Uniface field into a double.

Syntax

```
int DO_PSV_uf2dbl(UNIFACE_ENTITY *pentity, char *fieldname, double *pdbl);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field.
pdbl	A pointer to a double variable that will receive the value.

DO_PSV_uf2long

Converts a Uniface field into a long.

Syntax

```
int DO_PSV_uf2long(UNIFACE_ENTITY *pentity, char *fieldname, long *plong);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field.
plong	A pointer to a long that will receive the value.

Example

```
long lValue;
...
DO_PSV_uf2long(ONE_ENTITY_PKTEST_SZ1, "PK", &lValue);
RR_printf("PK :%d\n", lValue);
```

DO_PSV_uf2str

Converts a Uniface field into a C null terminated string.

Syntax

```
int DO_PSV_uf2str(UNIFACE_ENTITY *pentity, char *fieldname, char *dst, int dsize);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.

- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
fieldname	The name of the field.
dst	A pointer to a buffer that receives the string.
dsize	The size of the buffer that receives the string.

Example

```
#define STR_VALUE_LEN 30
char szValue[STR_VALUE_LEN+1];

...

DO_PSV_uf2str(ONE_ENTITY_PKTEST_SZ1, "F1", szValue, STR_VALUE_LEN );
RR_printf("F1 : %s\n",szValue);
```

DO_PSV_update

Updates the current record.

Syntax

```
int DO_PSV_update(UNIFACE_ENTITY *pentity, char *mode, char *hitstatus);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.

- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure representing the table of the record to be updated.
mode	The mode of execution (0000 always zero).

Example

```

int rrobot_script(s_info)
PLAYER_INFO *s_info;

{
...

POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_si1, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_si1);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1", "ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_SI1, "00FE" );

...

DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_creocc(ONE_ENT_PKTEST);
DO_PSV_str2uf(ONE_ENT_PKTEST, "PK","9");

/* read
*
* options
*/

DO_PSV_select(ONE_ENT_PKTEST, "0001", "000A", "0200", "", "", "");
DO_PSV_setocc(ONE_ENT_PKPKTEST, 1);
DO_PSV_fetch(ONE_ENT_PKTEST, "0001", "0046");
DO_PSV_close($DEF_UNIFACE_SI1, "0000", $DEF_UNIFACE_SI1,
ONE_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_write

Inserts a new record.

DO_PSV_write inserts a new record in the table specified pentity. The new record is the current occurrence of the table described by pentity.

Syntax

```
int DO_PSV_write(UNIFACE_ENTITY *pentity, char *mode);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.
mode	The mode of execution (0000 always zero).

Example

```
int rrobot_script(s_info)
PLAYER_INFO *s_info;
{
...
POLY_PROTOCOL PROTO;
LOGON_PATH PATH_1;

BEGIN_UENTITY($def_uniface_si1, "$DEF", "UNIFACE", "SI1", "$DEF,UNIFACE,SI1")
END_UENTITY($DEF_UNIFACE_SI1, $def_uniface_si1);

BEGIN_UENTITY(one_ent_pktest_sz1, "ONE_ENT", "PKTEST",
"SZ1", "ONE_ENT,PKTEST,SZ1,PK,N2,M,10,100,2,1.101,F1,S 2,0,100,20,")
UFIELD("PK", "N2", 2, "1.101")
UFIELD("F1", "S2", 20, "")
END_UENTITY(ONE_ENT_PKTEST_SZ1, one_ent_pktest_sz1);

DO_PSV_init(&PROTO, POLY_NET_TCP, s_info, 0);
```

```

BEGIN_TRANSACTION();
DO_PSV_logon(&PROTO, &PATH_1, "myhost+12000", "user", "password", "PSV1");
DO_PSV_open(&PATH_1, $DEF_UNIFACE_S11, "00FE" );
...
DO_PSV_open(&PATH_1, ONE_ENT_PKTEST_SZ1, "00C8" );
DO_PSV_modify(ONE_ENT_PKTEST_SZ1, "0000 ! 0002000");
DO_PSV_creocc(ONE_ENT_PKTEST);
DO_PSV_str2uf(ONE_ENT_PKTEST, "PK", "9");
DO_PSV_str2uf(ONE_ENT_PKTEST, "F1", "REC");
DO_PSV_write(ONE_ENT_PKTEST, "0000");
{script body ...}
...
DO_PSV_close($DEF_UNIFACE_S11, "0000", $DEF_UNIFACE_S11,
ONE_ENT_PKTEST_SZ1, NULL);
DO_PSV_logoff(&PATH_1);
END_TRANSACTION();

DO_PSV_clean(&PROTO);
REPORT(SUCCESS);
EXIT();
return(0);
}

```

DO_PSV_xtrans

Sends an X transaction to a polyserver.

Syntax

```
int DO_PSV_xtrans(UNIFACE_ENTITY *pentity, char *mode, UNIFACE_ENTITY *pentity, ...);
```

Return Value

0 = success

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure.

mode	Mode to send.
pentity	A pointer to a UNIFACE_ENTITY to transaction is sent to.

DO_PSV_zero

Sends a 0 transaction to a polyserver.

Syntax

```
int DO_PSV_zero(UNIFACE_ENTITY *pentity, char *extrainfo);
```

Return Value

0 = successful

< 0 = error. See the error descriptions that follow:

- ! 201 Unknown protocol specified.
- ! 202 An error was detected in the returned transaction. This error is sent to the Player as a warning, since it may be normal.
- ! 203 Invalid length of a parameter.
- ! 204 A unexpected NULL parameter was passed to a call.
- ! 205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
- ! 206 Invalid field name.
- ! 207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
- ! 401 Invalid token found while parsing where clause or aggregate statements.
- ! 501 No hit.

Parameters

Parameter	Description
pentity	A pointer to a UNIFACE_ENTITY structure
extrainfo	Binary information to be sent to the polyserver process.

Winsock

Winsock index

AddrByte

Returns a byte of an internet address.

DO_WSK_Accept

Accepts an incoming connection on the specified socket.

DO_WSK_Bind

Associates a local name with a connection/socket that is not yet named.

DO_WSK_Closesocket

Closes the specified connection.

DO_WSK_Connect

Establishes a connection with a server.

DO_WSK_Expect

Waits for a unique pattern to occur that should signify the end of the response.

DO_WSK_ExpectAny

Waits for any of the specified patterns to be matched.

DO_WSK_ExpectAnyExpr

DO_WSK_ExpectAnyExpr() waits for any of the unique patterns specified by the passed UNIX-style regular expressions to occur. The patterns should signify any of the possible ends of the response.

DO_WSK_ExpectExpr

DO_WSK_ExpectExpr() waits for a unique pattern specified by a UNIX-style regular expression to occur. The pattern should signify the end of the response.

DO_WSK_GetSocket

Returns the socket handle for the specified connection.

DO_WSK_Getsockname

Gets the local address for a connection.

DO_WSK_HexDecode

Converts hexadecimal characters to binary data suitable for sending to a connection using DO_WSK_Write().

DO_WSK_Init

Initializes internal structures and variables in preparation for a virtual user run.

DO_WSK_Ioctlsocket

Controls the mode of a socket.

DO_WSK_IsReadable

Specifies whether or not the connection has data available to be read.

DO_WSK_IsWritable

Indicates if the connection is available for writing.

DO_WSK_Listen

Puts the specified socket in listening mode for incoming connections.

DO_WSK_Quiet

Waits for a period of silence, identified by seconds_of_quiet, on the named socket.

DO_WSK_Read

Reads the number of bytes identified by bytes_to_read from the socket.

DO_WSK_Recv

Receives data from a connected socket.

[DO_WSK_Recvfrom](#)

Receives data from a connected or unconnected socket.

[DO_WSK_Reorder](#)

DO_WSK_Reorder() swaps the byte order of the given integer variable.

[DO_WSK_Select](#)

Allows you to determine if a set of sockets are read or writable.

[DO_WSK_Send](#)

Sends data to a socket.

[DO_WSK_SendAll](#)

Sends a number of strings to a connection.

[DO_WSK_Sendto](#)

Sends data on either a connected or unconnected socket to a remote host.

[DO_WSK_SetsockOpt](#)

Sets options associated with the specified socket.

[DO_WSK_Shutdown](#)

Disables the sending or receiving of data on a socket.

[DO_WSK_Socket](#)

Creates a socket and associates it with a connection handle.

[DO_WSK_Write](#)

Writes the number of bytes identified by bytes_to_write to the socket from data_to_send.

[EscapeStr](#)

Converts ^ and null characters into ^^ and ^@, respectively, so that data with those characters can be passed to DO_WSK_Send(), DO_WSK_Expect(), or DO_WSK_ExpectAny().

[GetLocalAddr](#)

Returns the local address used by a connection in host-byte order.

[GetLocalPort](#)

Returns the port bound to for the named socket on the local side of the connection.

[GetRemoteAddr](#)

Returns the port connected to on the remote side of a connection.

[GetRemotePort](#)

Returns the port connected to on the remote side of a connection.

[Getsockname](#)

Is called to get the local address and port for the connection.

[HiByte](#)

Returns the high-order byte of the passed short integer.

[LoByte](#)

Returns the low-order byte of the passed short integer.

Log

Records the character string passed into the log file.

MyByteOrder

MyByteOrder() returns the byte order of the machine running the script either of the constants MSBF (Most Significant Byte First) or LSBF (Least Significant Byte First).

Response

Returns a pointer to the first character in the response buffer.

ResponseLength

Returns the number of characters in the response buffer.

ScanExpr

Scans the scan buffer for a string specified by the UNIX-style regular expression, into the given buffer.

ScanFloat

Scans a floating point value of the given byteorder and length into the argument which should be the address of an appropriate program variable of the same size and type, casted to a char *. Valid lengths are 4 or 8. The byteorder should be either specified as either of the constants MSBF or LSBF.

ScanInt

ScanInt() scans an integer of the given byteorder and length into the argument which should be the address of an appropriate program variable of the same size and type, casted to a char *. Valid lengths are 1, 2, or 4. The byteorder should be either specified as one of the constants MSBF or LSBF.

ScanLenString

ScanLenString() expects input of the format [count][string] where length is an integer of the given byteorder and length and string is a string of count bytes. The string will be placed in the given pointer and count, which should be the address of an appropriate integral program variable, casted to a char *, and to be updated with the count.

ScanRewind

Resets the scan pointer and length to the beginning and length of the response buffer respectively.

ScanSkip

Skips the specified number of bytes in the scan buffer.

ScanString

Scans a string of the given length from the current location in the scan buffer into the given buffer. The scan pointer and length are incremented by the argument length.

SetTimeout

Sets the number of seconds to wait for subsequent synchronization commands (DO_WSK_Expect, DO_WSK_ExpectAny, or DO_WSK_Read) to be satisfied.

SetTypeRate

Sets the type rate, in characters per second, for data sent on a Telnet connection.

SkipExpr

Scans the scan buffer for a string specified by the UNIX-style regular expression, and skips ahead over the matched pattern.

UnEscapeStr

QALoad 5.02

Converts a string with escaped ^ control character sequences to raw text so that it can be manipulated.

QALoad character encoding

QALoad encodes non-printable ASCII characters (characters below ASCII 32 or above ASCII 128) for all QALoad Winsock script commands, and the following QALoad WWW commands:

```
! DO_Http
! DO_Https
! DO_VerifyDocTitle
```

Encoding prevents string handling and manipulation problems. The following table presents the character encoding for all encoded 7-bit ASCII characters (less than ASCII 128/Octal 200).

Code	Octal	Char
^@	000	NUL
^A	001	SOH
^B	002	STX
^C	003	ETX
^D	004	EOT
^E	005	ENQ
^F	006	ACK
^G	007	BEL
^H (\b for WWW)	010	BS
\t	011	HT
\n	012	LF
^K	013	VT
\f	014	FF
\r	015	CR
^N	016	SO
^O	017	SI
^P	020	DLE
^Q	021	DC1
^R	022	DC2
^S	023	DC3

^T	024	DC4
^U	025	NAK
^V	026	SYN
^W	027	ETB
^X	030	CAN
^Y	031	EM
^Z	032	SUB
^[033	ESC
\034	034	FS
^]	035	GS
\036	036	RS
^_	037	US
\"	042	"
\\	134	\
^^	136	^
^?	177	DEL

ASCII characters 128 and higher are encoded by using “\000”, where 000 is the octal value of the ASCII character.

Encoded data appears in the QALoad Winsock capture file in hexadecimal format:
0951414c6f616420697320224772656174220d0a00

This same data appears in a user’s script as:
\t QALoad is \"Great\"\\r\\n^@

AddrByte

Returns a byte of an internet address.

It’s only useful in very specific instances — most particularly when scripting an FTP client which requires sending the address of the client-side data port as separate bytes.

AddrByte returns the byte of the passed address indicated by which_byte.

Syntax

```
unsigned char
AddrByte(unsigned long address, int which_byte)
```

Parameters

Parameter	Description
unsigned long address	The address of the client-side data port.

int which byte	The byte to send.
----------------	-------------------

[See Also](#)

GetRemotePort, GetLocalAddr, GetLocalPort, AddrByte, HiByte, LoByte

[Example](#)

```
unsigned char byte0, byte1, byte2, byte3;
...
byte0 = AddrByte(GetLocalAddr(S2), 0);
byte1 = AddrByte(GetLocalAddr(S2), 1);
byte2 = AddrByte(GetLocalAddr(S2), 2);
byte3 = AddrByte(GetLocalAddr(S2), 3);
```

[DO_WSK_Accept](#)

Accepts an incoming connection on the specified socket.

Returns a new socket handler if successful or -1 if an error occurs.

[Syntax](#)

```
SOCKET DO_WSK_Accept(int nSocketHandle, int newSocketHandle)
```

[Parameters](#)

Parameter	Description
nSocketHandle	Socket handle from a previous call to DO_WSK_Socket.
newSocketHandle	New Socket handle for accepting connections.

[Example](#)

```
DO_WSK_Accept(S1, S2 );
```

[DO_WSK_Bind](#)

Associates a local name with a connection/socket that is not yet named.

[Syntax](#)

```
DO_WSK_Bind (int nConnectHandle, char * szLocalInetAddr, unsigned short usPort);
```

[Parameters](#)

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
szLocalInetAddr	Points to a string containing the Internet address the socket is to bind to. For example, to bind to INADDR_ANY, szLocalInetAddr would point to "0.0.0.0".
usPort	The port to bind to in host byte order. To bind to any (non-specific) port, pass 0.

[Example](#)

```
DO_WSK_Bind (0, "0.0.0.0", 0);
```

DO_WSK_Closesocket

Closes the specified connection.

Syntax

```
DO_WSK_Closesocket (int nConnectHandle);
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.

Example

```
DO_WSK_Closesocket (0);
```

DO_WSK_Connect

Establishes a connection with a server.

Syntax

```
int DO_WSK_Connect (int nConnectHandle, char * szServerInetAddr, unsigned short usPort, int nAddressfamily);
```

Return Value

Returns 0 if the function call was successful or -1 if an error occurred.

Parameters

Parameter	Description
nConnectHandle	Connection handle returned from a previous call to DO_WSK_Socket.
szServerInetAddr	Points to a string containing the Internet address of the server with which to connect.
usPort	The port (in host-byte order) on the server with which to connect.
nAddressfamily	Address family, usually AF_INET.

Example

```
DO_WSK_Connect ( 0, "172.22.1.130", 53, 2 );
```

DO_WSK_Expect

Waits for a unique pattern to occur that should signify the end of the response.

When the capture file is converted, this pattern is identified automatically. If the response changes, the pattern may need to be adjusted or another synchronization command substituted in the place of DO_WSK_Expect().

DO_WSK_Expect returns 0 if the pattern was found or -1 if the timeout interval expired.

Syntax

```
int DO_WSK_Expect(int nConnectHandle, char *pattern)
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
pattern	A string pattern to wait for

Example

```
DO_WSK_Expect(S1, "\r\n");
```

DO_WSK_ExpectAny

Waits for any of the specified patterns to be matched.

DO_WSK_ExpectAny returns the 0-based index of the pattern that was matched first, or -1 if not found.

Syntax

```
int DO_WSK_ExpectAny(int nConnectHandle, int number_of_patterns, char *pattern1, ...)
```

Parameters

Parameter	Description
nConnectHandle	The connection number.
number_of_patterns	The number of patterns to follow.
pattern1	The first pattern.

Example

```
int i;
...
i = DO_WSK_ExpectAny(S1, 3, "this", "that", "the other");
switch(i)
{
case 0: /* do stuff because "this" was matched */ break;
case 1: /* do stuff because "that" was matched */ break;
case 2: /* do stuff because "the other" was matched */ break;
default: /* must have timed out */
}
```

DO_WSK_ExpectAnyExpr

DO_WSK_ExpectAnyExpr() waits for any of the unique patterns specified by the passed UNIX-style regular expressions to occur. The patterns should signify any of the possible ends of the response.

DO_WSK_ExpectAnyExpr() returns the index of the pattern that was matched (0-based) or -1 if the timeout interval expired.

Syntax

```
int DO_WSK_ExpectAnyExpr(int nConnectHandle, int num_expressions, char *expression1, char *expression2, ...)
```


Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
Expression1	String patterns to wait for.

Example

```
DO_WSK_ExpectAnyExpr(S1, 2, "query failed [0-9]* times", "query succeeded [0-9]* times");
```

DO_WSK_ExpectExpr

DO_WSK_ExpectExpr() waits for a unique pattern specified by a UNIX-style regular expression to occur. The pattern should signify the end of the response.

DO_WSK_ExpectExpr() returns 0 if the pattern was found or -1 if the timeout interval expired.

Syntax

```
int DO_WSK_ExpectExpr(int nConnectHandle, char *expression)
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
expression	A string pattern to wait for.

Example

```
DO_WSK_ExpectExpr(S1, "the date is [0-9][0-9]/[0-9][0-9]/[0-9][0-9]");
```

DO_WSK_GetSocket

Returns the socket handle for the specified connection.

This allows more complicated examples of determining if multiple sockets are available for writing or if data is available for reading using the select() system call.

Syntax

```
SOCKET DO_WSK_GetSocket(int ConnectHandle)
```

Parameters

Parameter	Description
ConnectHandle int	The connection number.

Example

```
SOCKET x = DO_WSK_GetSocket(S1);
SOCKET y = DO_WSK_GetSocket(S2);
fd_set readfds;
struct timeval timeout;
int maxfds;
timeout.tv_sec = 1;
```

QALoad 5.02

```
timeout.tv_usec = 0;
FD_SET(x, &readfds);
FD_SET(y, &readfds);
if(x > y) maxfds = x; else maxfds = y;
select(maxfds+1, &readfds, 0, 0, timeout);
Waits for 1 second for data to be available for reading on connection S1 or S2.
```

DO_WSK_Getsockname

Gets the local address for a connection.

Use this call or DO_WSK_Bind prior to a call to GetLocalAddr or GetLocalPort.

Syntax

```
unsigned short DO_WSK_Getsockname(int nConnectHandle)
```

Parameters

Parameter	Description
nConnectionHandle	A connection handle from a previous call to DO_WSK_Socket.

Example

```
DO_WSK_Socket(S1, AF_INET, SOCK_DGRAM, IPPROTO_UDP);
DO_WSK_Bind(S1, "127.0.0.1", ANY_PORT);
fd_set readfds;
DO_WSK_Getsockname(S1);
```

DO_WSK_HexDecode

Converts hexadecimal characters to binary data suitable for sending to a connection using DO_WSK_Write().

DO_WSK_HexDecode returns the number of bytes in the converted data (one half the number of input bytes).

Syntax

```
int HexDecode(char *string)
```

Parameters

Parameter	Description
string	A pointer to a buffer to be converted.

Example

```
char buf[80];
int count;

...

strcpy(buf, "FEEBDAED");
count = DO_WSK_HexDecode(buf);
DO_WSK_Write(S1, buf, count);
```

DO_WSK_Init

Initializes internal structures and variables in preparation for a virtual user run. DO_WSK_Init returns 1.

Syntax

```
int DO_WSK_Init(s_info)
```

Parameters

Parameter	Description
s_info	A pointer to the PLAYER_INFO structure for this virtual user.

Example

```
DO_WSK_Init (s_info);
```

DO_WSK_Ioctlsocket

Controls the mode of a socket.

Syntax

```
DO_WSK_Ioctlsocket(int nConnectHandle, unsigned long * argument, long command );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
argument	The parameter for command. If command is FIONBIO, and argument points to a non-zero value, non-blocking mode is enabled. If command is FIONREAD, argument is used to hold the number of bytes that can be read on a socket. If command is SIOCATMARK, argument is used as a return value to determine if all out-of-band data has been read from a socket. TRUE is returned if no out-of-band data is to be read.
command	The command to perform on the sockets. Valid values are FIONBIO, FIONREAD, and SIOCATMARK.

Example

```
// enable non-blocking mode
u_long argument = TRUE;
DO_WSK_ioctlsocket(0, &argument, FIONBIO );
```

DO_WSK_IsReadable

Specifies whether or not the connection has data available to be read.

Returns 1 if the connection has data available to be read. Returns 0 if there is no data available. Returns -1 if there was an error.

Syntax

```
int DO_WSK_IsReadable(int ConnectHandle);
```

Parameters

Parameter	Description
ConnectHandle int	The connection number.

Example

```
do
{
DO_WSK_Read(S1, 4);
}
while (DO_WSK_IsReadable(S1)) ;
//Reads 4 bytes of data at a time until there is no more data to be read.
```

DO_WSK_IsWriteable

Indicates if the connection is available for writing.

Returns 1 if the connection is available for writing. Returns 0 if the output queue is full. Returns -1 if there was an error.

Syntax

```
int DO_WSK_IsWriteable(int ConnectHandle);
```

Parameters

Parameter	Description
ConnectHandle int	The connection number.

Example

```
do
{
DO_SLEEP(1);
}
while (DO_WSK_IsWriteable(S1) == 0) ;
DO_WSK_Send(S1, "stuff");
//Waits until connection S1 is writable before sending "stuff".
```

DO_WSK_Listen

Puts the specified socket in listening mode for incoming connections.

DO_WSK_Listen always returns 0.

Syntax

```
int DO_WSK_Listen(int nSocket);
```

Parameters

Parameter	Description
nSocket	Socket handle from a previous call to DO_WSK_Socket.

Example

```
DO_WSK_Listen(S1);
```

DO_WSK_Quiet

Waits for a period of silence, identified by `seconds_of_quiet`, on the named socket.

This can be useful if the response is random or you simply don't know what the response will be.

`DO_WSK_Quiet()` will read whatever characters are available. After characters are read, the `seconds_of_quiet` counter is reset. If a socket is not idle, `DO_WSK_Quiet` cannot complete.

`DO_WSK_Quiet` returns the number of bytes read, or -1 if an error was encountered.

Syntax

```
int DO_WSK_Quiet(int nConnectHandle, double seconds_of_quiet)
```

Parameters

Parameter	Description
<code>nConnectHandle</code>	The connection number.
<code>seconds_of_quiet</code>	The number of seconds of silence to wait for on this connection.

Example

```
DO_WSK_Quiet(S2, 10);
```

DO_WSK_Read

Reads the number of bytes identified by `bytes_to_read` from the socket.

This can be useful if the response varies in content, but the number of bytes is consistent. It can also be useful to build scripts that handle more complicated protocols.

`DO_WSK_Read` returns the number of bytes read, or -1 if an error was encountered.

Syntax

```
int DO_WSK_Read(int nConnectHandle, int bytes_to_read)
```

Parameters

Parameter	Description
<code>nConnectHandle</code>	The connection number.
<code>bytes_to_read</code>	The number of bytes to wait for.

Example

```
DO_WSK_Read(S2, 1024);
```

DO_WSK_Recv

Receives data from a connected socket.

Syntax

```
DO_WSK_Recv (int nConnectHandle, char * buffer, int * buffer_length, int flags, int * pnBytesRecv)
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
buffer	Buffer to store received data.
buffer_length	Length of buffer.
flags	Used to control the way in which the call is made. Valid values are MSG_PEEK, MSG_OOB, or 0.
pnBytesRecv	Used to return the number of bytes received.

Example

```
char buffer[1024];
int nBytesReceived;
DO_WSK_Recv (0, buffer, 1024, 0, &nBytesReceived);
```

DO_WSK_Recvfrom

Receives data from a connected or unconnected socket.

Syntax

```
DO_WSK_Recvfrom (int nConnectHandle, char * buffer, int buffer_length, int flags, struct
sockaddr *from_address, int * pnBytesRecv );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
buffer	Buffer to store received data.
buffer_length	Length of buffer.
flags	Used to control the way in which the call is made. Valid values are MSG_PEEK, MSG_OOB, or 0.
from_address	Optional. If not NULL, it is used to hold the address of the sender upon function return.
pnBytesRecv	Used to return the number of bytes received.

Example

```
char buffer[1024];
int nBytesReceived;
DO_WSK_Recvfrom(0, buffer, 1024, 0, NULL, &nBytesReceived);
```

DO_WSK_Reorder

DO_WSK_Reorder() swaps the byte order of the given integer variable.

DO_WSK_Reorder() returns nothing.

Syntax

```
void DO_WSK_Reorder(int size, void *value)
```

Parameters

Parameter	Description
size	Size of the integer variable (1, 2, or 4 bytes).
value	The address of the variable.

Example

```
int var;
var = 2;
DO_WSK_Reorder(sizeof(int), (char *)&var);
DO_WSK_Send(S2, EscapeStr((char *)&var, sizeof(int)));
```

DO_WSK_Select

Allows you to determine if a set of sockets are read or writable

Syntax

```
Int DO_WSK_Select(fd_set *readfds, fd_set *writefds, fd_set *selectfds, struct timeval *timeout);
```

Parameters

Parameter	Description
readfds	Set of sockets (struct fd_set) to check for read.
writefds	Set of sockets (struct fd_set) to check for write.
selectfds	Set of sockets (struct fd_set) to check for errors.
timeout	Maximum time for select to wait using timeval struct.

Example

```
fd_set *Set1 = malloc(sizeof(fd_set));
fd_set *Set2 = malloc(sizeof(fd_set));
fd_set *Set3 = malloc(sizeof(fd_set));
struct timeval *Time = malloc(sizeof(fd_set));

....

....

DO_WSKk_Socket(S3, AF_INET, SOCK_STREAM, IPPROTO TCP);
DO_WSK_Bind(S3, ANY_ADDR, ANY_PORT);
DO_WSK_Getsockname(S3);
DO_WSK_Connect(S3, "172.22.11.25", 80, AF_INET);

Set1->fd_count = 1;
Set2->fd_count = 1;
Set3->fd_count = 1;
Set1->fd_array[0] = S3;
Set2->fd_array[0] = S3;
Set3->fd_array[0] = S3;
Time->tv_sec = 1;
DO_WSK_Select(Set1, Set2, Set3, Time);
```

QALoad 5.02

```
free(Set1);  
free(Set2);  
free(Set3);  
free(Time);
```

DO_WSK_Send

Sends data to a socket.

DO_WSK_Send returns 0 if successful.

Syntax

```
DO_WSK_Send(int nConnectHandle, char *data)
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
data	Data to be sent.

Example

```
DO_WSK_Send(S1, "This is sent to connection S1");
```

DO_WSK_SendAll

Sends a number of strings to a connection.

Syntax

```
DO_WSK_SendAll(int nConnectHandle, int numstrings, char string1, char *string2, ...);
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
numstrings	The number of strings to be sent.
string1	The strings to be sent, separated by commas.

Example

```
DO_WSK_Socket(S3, AF_INET, SOCK_STREAM, IPPROTO_TCP);  
DO_WSK_Bind(S3, ANY_ADDR, ANY_PORT);  
DO_WSK_Getsockname(S3);  
DO_WSK_Connect(S3, "172.22.11.25, 80, AF_INET);  
DO_WSK_Setsockopt(S3, IPPROTO_TCP, TCP_NODELAY, 1);  
DO_WSK_Setsockopt(S3, SOL_SOCKET, SO_LINGER, 0x100);  
DO_WSK_SendAll(S3, 2, "GET/HTTP/1.1\r\nAccept: image/gif,"  
"image/x-xbitmap, image/jpeg, image/pjpeg, application/"  
"vnd.ms-excel, application/msword, application/x-shock"  
"wave-flash, */*\r\nAccept-Language: en-us\r\nAccept-En"  
"coding: gzip, deflate\r\nIf-Modified-Since: Mon, 03 Feb"
```



```
"2003 15:03:15 GMT\r\nIf-None-Match:\r\n82f2e16095cbc21:"
"973\r\n", "\r\nUser-Agent: Mozilla/4.0 (compatible; MSIE"
"6.0; Windows NT 5.0; .NET CLR 1.0.3705)\r\nHost:"
"qaappserv\r\nConnection: Keep-Alive\r\n\r\n");
```

DO_WSK_Sendto

Sends data on either a connected or unconnected socket to a remote host.

Syntax

```
Int DO_WSK_Sendto (int nConnectHandle, char * wsk_statement, char szServerInetAddr, unsigned
short port );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
wsk_statement	Buffer of data to be sent.
szServerInetAddr	Character string containing the Internet address of the destination socket.
unsigned short port	The port (in host-byte order) of the destination socket.

Example

```
DO_WSK_Socket(S1, AF_INET, SOCK_DGRAM, IPPROTO_IP);
DO_WSK_Setsockopt(S1, SOL_SOCKET, SO_BROADCAST, 1);
DO_WSK_Bind(S1, ANY_ADDR, ANY_PORT);
DO_WSK_Sendto(S1, "0$^B^A^@^D^Fpublic\241^W^B^A^A^B^A^@^B^A^@0\f0\n^F^F+^F^"
"A^B^A^A^E^@", "172.22.6.71", 161);
```

DO_WSK_Setsockopt

Sets options associated with the specified socket.

Syntax

```
DO_WSK_Setsockopt(int nConnectHandle, int level, int option_name, int wsk_sockopt_optval );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
level	The level at which the socket option is defined. This can be either SOL_SOCKET or IPPROTO_TCP.
option_name	Option name. Refer to your Winsock documentation for a complete list of values.
wsk_sockopt_optval	Integer variable will receive the values of option_name upon function return.

Example

```
DO_WSK_Connect(S3, "172.22.11.25", 80, AF_INET);
DO_WSK_Setsockopt(S3, IPPROTO_TCP, TCP_NODELAY, 1);
DO_WSK_Setsockopt(S3, SOL_SOCKET, SO_LINGER, 0x100);
```

DO_WSK_Shutdown

Disables the sending or receiving of data on a socket.

Syntax

```
DO_WSK_Shutdown (int nConnectHandle, int shutdown_type );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
shutdown_type	If equal to 0, all receives are disabled. If equal to 1, all sends are disabled. If equal to 2, all sends and receives are disabled.

Example

```
// disable sends
DO_WSK_Shutdown (0, 1);
```

DO_WSK_Socket

Creates a socket and associates it with a connection handle.

Syntax

```
DO_WSK_Socket (int nConnectHandle , int address_family, int type, int protocol );
```

Parameters

Parameter	Description
nConnectHandle	A connection handle to associate with a new socket.
address_family	The address family the socket will use. Refer to your Winsock documentation for a complete list.
type	Either SOCK_DGRAM for UDP or SOCK_STREAM for TCP.
protocol	Specifies which protocol will be used with the socket. Refer to your Winsock documentation for a complete list of protocols.

Example

```
// create a stream socket
DO_WSK_Socket (0, AF_INET, SOCK_STREAM, 0 );
```

DO_WSK_Write

Writes the number of bytes identified by bytes_to_write to the socket from data_to_send.

This can be used in place of `DO_WSK_Send()` when coding scripts by hand. `DO_WSK_Send()` expects a string that has certain control and null characters encoded. `DO_WSK_Write` does not expect any encoding, and so can be used to send data without having to use `EscapeStr` to encode any possible control characters. `DO_WSK_Write` returns the number of bytes written, or -1 if an error was encountered.

Syntax

```
int DO_WSK_Write(int nConnectHandle, char *data_to_send, int bytes_to_write)
```

Parameters

Parameter	Description
nConnectHandle	The connection number.
data_to_send	The data to write to the connection.
bytes_to_write	The number of bytes to write.

See Also

[DO_WSK_Send](#), [EscapeStr](#), [UnEscapeStr](#)

Example

```
DO_WSK_Write(S2, buffer, 1024);
```

EscapeStr

Converts ^ and null characters into ^^ and ^@, respectively, so that data with those characters can be passed to `DO_WSK_Send()`, `DO_WSK_Expect()`, or `DO_WSK_ExpectAny()`.

`EscapeStr` returns a pointer to the converted characters.

Syntax

```
char * EscapeStr(char *string, int count)
```

Parameters

Parameter	Description
string	A pointer to a buffer to be converted.
count	The number of characters to convert.

Example

```
char buf[80];
...
DO_WSK_Send(S1, EscapeStr("\0\0\0x^hello", 10));
```

GetLocalAddr

Returns the local address used by a connection in host-byte order.

This can be useful in any case where the client application (and hence, the script) uses `DO_WSK_Bind()` to bind a socket to an unspecified address and/or port and then does a `DO_WSK_Listen()` on that socket. This

sequence indicates that the application tells the remote side what the local address and port are so that it can connect back to the application (identified by a `DO_WSK_Accept()`). In this case, it is necessary to identify how the client application is informing the remote application of what address and port it is listening on.

For example: The ftp client application binds to the first available port and does a `listen()` on that port. The client tells the remote side, which is actually the ftp server, what port it is listening on by sending a command that looks like "PORT 172,23,70,242,4,212\r\n" where 172,23,70,242 is the IP address of the local machine and 4,212 are the high-order byte and low-order byte of the port number being listened on. To make this slightly easier, we've included the `HiByte()`, `LoByte()`, and `AddrByte()` functions also documented in this chapter.

`GetLocalAddr` returns the address of the local side of the connection.

Syntax

```
unsigned long GetLocalAddr(int nConnectHandle)
```

Parameters

Parameter	Description
<code>nConnectHandle</code>	The connection number.

Example

```
unsigned long addr;
unsigned short port;

...

/*
the following lines have to be AFTER socket S3 is bound in a DO_WSK_Bind() call
port = GetLocalPort(S3);
addr = GetLocalAddr(S3); /* now we know both the local address and port */

...

{
char buf[80];
sprintf(buf, "PORT %d,%d,%d,%d,%d,%d\r\n",
AddrByte(addr,0),
AddrByte(addr,1),
AddrByte(addr,2),
AddrByte(addr,3),
HiByte(port),
LoByte(port));
DO_WSK_Send(S4, buf);
}
```

GetLocalPort

Returns the port bound to for the named socket on the local side of the connection.

Syntax

```
unsigned short GetLocalPort(int nConnectHandle)
```

Parameters

Parameter	Description
<code>nConnectHandle</code>	The connection number.

Example

```
unsigned short port;
...
port = GetLocalPort(S3);
```

GetRemoteAddr

Returns the port connected to on the remote side of a connection.

GetRemoteAddr returns the address of the remote side of the connection as a long integer.

Syntax

```
unsigned long GetRemoteAddr(int nConnectHandle)
```

Parameters

Parameter	Description
nConnectHandle	The connection number.

Example

```
unsigned long addr;
...
addr = GetRemoteAddr(S3);
```

GetRemotePort

Returns the port connected to on the remote side of a connection.

Syntax

```
unsigned short GetRemotePort(int nConnectHandle)
```

Parameters

Parameter	Description
nConnectHandle	The connection number.

Example

```
unsigned short port;
...
port = GetRemotePort(S3);
```

Getsockname

Is called to get the local address and port for the connection.

Either it, or DO_WSK_Bind must be called prior to a call to GetLocalAddr or GetLocalPort. Getsockname returns 0 for success.

Syntax

```
unsigned short Getsockname(int nConnectHandle)
```

Parameters

Parameter	Description
nConnectHandle	The connection number.

Example

```
Getsockname(S3);
```

HiByte

Returns the high-order byte of the passed short integer.

It's only useful in very specific instances — most particularly when scripting an FTP client that requires sending the high and low order bytes of the client-side data port as separate bytes.

HiByte returns the high-order byte of the passed short.

Syntax

```
unsigned char HiByte(short port)
```

Parameters

Parameter	Description
port	The short value whose high-order byte is being returned.

Example

```
unsigned char hbyte;
...
hbyte = HiByte(GetLocalPort(S3));
```

LoByte

Returns the low-order byte of the passed short integer.

It's only useful in very specific instances — most particularly when scripting an FTP client that requires sending the high and low order bytes of the client-side data port as separate bytes.

LoByte returns the low-order byte of the passed short.

Syntax

```
unsigned char LoByte(short port)
```

Parameters

Parameter	Description
port	The short value to return the low byte of.

Example

```
unsigned char lbyte;
...
byte = LoByte(GetLocalPort(S3));
```

Log

Records the character string passed into the log file.

Syntax

```
void Log(char *line_to_be_logged)
```

Parameters

Parameter	Description
line_to_be_logged	A string to be written to the log file

Example

```
Log("Got here!");
```

MyByteOrder

MyByteOrder() returns the byte order of the machine running the script – either of the constants MSBF (Most Significant Byte First) or LSBF (Least Significant Byte First).

MyByteOrder() returns the byteorder of the machine running the script.

Syntax

```
int MyByteOrder()
```

Parameters

None.

Example

```
short value;
int myorder = MyByteOrder();
...
ScanInt(myorder, 2, (char *)&value);
```

Response

Returns a pointer to the first character in the response buffer.

It should be called immediately after a DO_WSK_Expect(), DO_WSK_ExpectAny(), DO_WSK_Read(), or DO_WSK_Quiet().

Response returns a pointer to the matched response.

Syntax

```
char * Response()
```

Parameters

None.

Example

```
char buf[80];
...
sprintf(buf, "The first 10 characters of the response
were %10.10s",Response());
Log(buf);
```

ResponseLength

Returns the number of characters in the response buffer.

Syntax

```
int ResponseLength()
```

Parameters

None.

Example

```
char buf[80];
...
sprintf(buf, "The length of the response was %d bytes",
ResponseLength());
Log(buf);
```

ScanExpr

Scans the scan buffer for a string specified by the UNIX-style regular expression, into the given buffer. ScanExpr() returns the number of bytes matched by the expression.

Syntax

```
int ScanExpr(char *exprstr, char *buffer)
```

Parameters

Parameter	Description
exprstr	The UNIX-style regular expression to look for.
buffer	Where to copy the string to.

Example

```
char buffer[80];
...
SkipExpr("the name is ");
ScanExpr(".*!", buffer);
Log("the name was %s", buffer);
```


ScanFloat

Scans a floating point value of the given byteorder and length into the argument which should be the address of an appropriate program variable of the same size and type, casted to a char *. Valid lengths are 4 or 8. The byteorder should be either specified as either of the constants MSBF or LSBF.

ScanFloat returns nothing.

Syntax

```
void ScanFloat(int byteorder, int length, char *buffer)
```

Parameters

Parameter	Description
byteorder	The byteorder of the floating point value.
length	The size of the floating point value (4 (float) or 8 (double)).
buffer	Where to copy the bytes to.

Example

```
float v1;
double v2;
...
ScanFloat(MyByteOrder(), sizeof(float), (char *)&v1);
ScanFloat(MyByteOrder(), sizeof(double), (char *)&v2);
Log("the v1 was %d and v2 was %d", v1, v2);
```

ScanInt

ScanInt() scans an integer of the given byteorder and length into the argument which should be the address of an appropriate program variable of the same size and type, casted to a char *. Valid lengths are 1, 2, or 4. The byteorder should be either specified as one of the constants MSBF or LSBF.

ScanInt returns nothing.

Syntax

```
void ScanInt(int byteorder, int length, char *buffer)
```

Parameters

Parameter	Description
byteorder	The byteorder of the integer value.
length	The size of the integer (1, 2, or 4).
buffer	Where to copy the bytes to.

Example

```
short port;
int length;
ScanInt(MyByteOrder(), sizeof(short), (char *)&port);
```

QALoad 5.02

```
ScanInt(MyByteOrder(), sizeof(int), (char *)&length);  
Log("the port was %d and the length was %d", port, length);
```

ScanLenString

`ScanLenString()` expects input of the format `[count][string]` where `length` is an integer of the given `byteorder` and `length` and `string` is a string of `count` bytes. The string will be placed in the given pointer and `count`, which should be the address of an appropriate integral program variable, type-cast to a `char *`, and to be updated with the count.

`ScanLenString()` returns nothing.

Syntax

```
void ScanLenString(int byteorder, int length, char *count, char *buffer)
```

Parameters

Parameter	Description
<code>byteorder</code>	The byteorder of the count value.
<code>length</code>	The size of the count value (1, 2, or 4).
<code>Count</code>	The address of a integral program value of the appropriate size to copy the count value to.
<code>buffer</code>	Where to copy the string to.

Example

```
int len;  
char buffer[80];  
...  
ScanLenString(MyByteOrder(), 4, (char *)&len, buffer);  
buffer[len] = '\\0';  
Log("the name was %s, length was %d", buffer, len);
```

ScanRewind

Resets the scan pointer and length to the beginning and length of the response buffer respectively.

`ScanRewind()` returns nothing.

Syntax

```
void ScanRewind( )
```

Parameters

None.

See Also

[ScanSkip](#), [ScanString](#), [ScanInt](#), [ScanFloat](#), [ScanLenString](#)

Example

```
ScanRewind( );
```

ScanSkip

Skips the specified number of bytes in the scan buffer.

ScanSkip() returns nothing.

Syntax

```
void ScanSkip(int count)
```

Parameters

Parameter	Description
Count	The number of bytes to skip ahead in the scan buffer.

Example

```
ScanSkip(5);
```

ScanString

Scans a string of the given length from the current location in the scan buffer into the given buffer. The scan pointer and length are incremented by the argument length.

ScanString() returns nothing.

Syntax

```
void ScanString(int length, char *buffer)
```

Parameters

Parameter	Description
length	The number of bytes to copy.
buffer	Where to copy the bytes to.

Example

```
char mybuf[6];
...
ScanString(5, mybuf);
mybuf[5] = '\0';
Log("the name was %s", mybuf);
```

SetTimeout

Sets the number of seconds to wait for subsequent synchronization commands (DO_WSK_Expect, DO_WSK_ExpectAny, or DO_WSK_Read) to be satisfied.

The default timeout value is 20 seconds. Increase this value for large user tests.

SetTimeout returns the previous timeout value.

Syntax

```
int SetTimeout(int seconds)
```

Parameters

Parameter	Description
seconds	The number of seconds to wait for a pattern in an <code>DO_WSK_Expect</code> or <code>DO_WSK_ExpectAny</code> , a connection in a <code>DO_WSK_Accept</code> , or a number of bytes to be received in a <code>DO_WSK_Read</code> .

Example

```
SetTimeout(20);
```

SetTyperate

Sets the type rate, in characters per second, for data sent on a Telnet connection.

Syntax

```
int SetTyperate(int count);
```

Parameters

Parameter	Description
count	The number of characters per second that are sent on the Telnet connection.

Example

```
SetTyperate(50);
```

SkipExpr

Scans the scan buffer for a string specified by the UNIX-style regular expression, and skips ahead over the matched pattern.

`SkipExpr()` returns the number of bytes matched by the expression.

Syntax

```
void SkipExpr(char *exprstr)
```

Parameters

Parameter	Description
exprstr	The UNIX-style regular expression to look for.

UnEscapeStr

Converts a string with escaped ^ control character sequences to raw text so that it can be manipulated.

`UnEscapeStr` returns the length of the converted string.

Syntax

```
int UnEscapeStr(char *string)
```

Parameters

Parameter	Description
string	A string with ^ control character sequences.

Example

```
char buf[80], str[6];
int len, x, y;

...

strcpy(buf, "^A^B^B^@hello\n");
len = UnEscapeStr(buf);
memcpy(&x, &buf[0], 2);
memcpy(&y, &buf[2], 2);
memcpy(str, &buf[4], 6);
```

WWW

WWW Index

Attach

Applies to Visual Scripting. Changes the current page to the page or frame specified. This new page becomes the active page that all Web functions act on.

Clear

Applies to Visual Scripting. Used to clear out certain items such as the cache or cookies. Types can be ordered together to clear both.

Click_On

Applies to Visual Scripting. Mimics the user clicking on text links, clickable images, and submit buttons.

DisableStatisticsRP

Applies to Real Networks Streaming Media. Disables capture of statistics during a load test.

DO_AddHeader

Applies to HTTP and SSL requests. Indicates headers that are common to every request (DO_Http or DO_Https function calls) in a script.

DO_AdditionalSubRequest

Applies to HTTP and SSL requests. DO_AdditionalSubRequest manually adds a sub-request for the next DO_Http or DO_Https request. The request is specified as a URL.

DO_AllowTrafficFrom

Applies to HTTP and SSL requests. If DO_AllowTrafficFrom is present in a script, then sub-requested URLs will only occur if the sub-request's URL contains one of the sub-strings in the substrings' list.

DO_AttachFile

Applies to HTTP and SSL requests. Specifies files that should be loaded into memory at the beginning of a script run. This is used in Post transactions that include binary files.

DO_AutomaticSubRequests

Applies to HTTP requests. Indicates whether subrequests will be downloaded during replay.

[DO_BasicAuthorization](#)

Specifies the username and password to gain access to a password protected WWW host, directory, or file.

[DO_BlankOutOfRangeData](#)

Applies to HTTP and SSL requests. If DO_BlankOutOfRangeData is enabled, then characters in the HTTP response body which interface with text searching or the HTML parser are changed to spaces.

[DO_BlockTrafficFrom](#)

Applies to HTTP and SSL requests. If DO_BlockTrafficFrom is present in a script, then sub-requested URLs will only occur if the sub-request's URL does not contain one of the sub-strings in the substrings' list.

[DO_Cache](#)

Applies to HTTP and SSL requests. Turns on cache emulation which caches anything with a content type beginning with "image/".

[DO_Clear](#)

Applies to HTTP and SSL requests. Used to clear out certain items such as the cache or cookies. Types can be ordered together to clear both.

[DO_ClearCache](#)

Applies to HTTP and SSL requests. Clears any cached images. Performed automatically by DO_HttpCleanup.

[DO_ClearDNSCache](#)

Applies to HTTP and SSL requests. When DO_Http or DO_Https make an HTTP request, DO_ClearDNSCache caches any DNS lookups that are performed. If that cache needs to be cleared to simulate browser, use DO_ClearDNSCache.

[DO_ClearJavascript](#)

Applies to HTTP and SSL requests. Clears any memory allocated by the JavaScript engine. Performed automatically by DO_HttpCleanup. DO_ClearJavascript is the same as DO_Clear (JAVASCRIPT_ENGINE).

[DO_DynamicCookieHandling](#)

Applies to HTTP and SSL requests. Turns dynamic cookie handling on or off.

[DO_DynamicRedirectHandling](#)

Applies to HTTP requests. Retrieves a redirected URL for use in the next request.

[DO_EnableJavascript](#)

Applies to HTTP requests. Enables or disables the interpretation of Javascript.

[DO_EncodeString](#)

Applies to HTTP and SSL requests. DO_EncodeString takes in a string and URL-encodes the string to be suitable to use as a CGI parameter or in the body of a POST.

[DO_FreeHttp](#)

Applies to HTTP and SSL requests. Also applies to Visual Scripting. Clears memory used by the script.

[DO_GetAnchorByNumber](#)

Applies to HTTP and SSL requests. Stores the value of an anchor from an HTML reply into a string that can be substituted into subsequent requests.

[DO_GetAnchorCount](#)

Applies to HTTP and SSL requests. Returns the total number of the anchors on the page.

[DO_GetAnchorHREF](#)

Applies to HTTP and SSL requests. Stores the value of a named anchor off of an HTML reply into a string that can be substituted into subsequent requests.

[DO_GetAnchorHREFEx](#)

Applies to HTTP and SSL requests. Stores the value of a specific occurrence of a named anchor off an HTML reply into a string that can be substituted into subsequent requests.

[DO_GetAnchorHREFn](#)

Applies to HTTP and SSL requests. Stores the value of a specific occurrence of a named anchor off of an HTML reply into a string that can be substituted into subsequent requests.

[DO_GetClientMapHREF](#)

Applies to HTTP and SSL requests. DO_GetClientMapHREF is used to extract the href URL from a particular region of a client-side image map. Client-side image maps are specified within an HTML document by the map' tag. Inside the map' tag, a' and area' tags are used to specify regions of the image map. The href attribute of the a' or area' tags specify the location of the URL to go to.

[DO_GetCookie](#)

Applies to HTTP and SSL requests. Extracts a cookie from the QALoad internal cookie list. The cookie is retrieved based on the name of the cookie. Wildcard patterns can be used to specify the cookie name in case the cookie name is dynamic. A count is also specified in case multiple cookies match the specified name.

[DO_GetCookieFromReplyEx](#)

Applies to HTTP and SSL requests. Retrieves and stores the value of a cookie when a Set-Cookie: statement is encountered in a reply header.

[DO_GetFormActionStatement](#)

Applies to HTTP and SSL requests. Gets the ACTION tag from a requested form. This feature is useful when a form dynamically changes what is stored in the ACTION tag.

[DO_GetFormValueByName](#)

Applies to HTTP and SSL requests. Retrieves the value embedded in a form for the specified field.

[DO_GetHeaderFromReply](#)

Applies to HTTP and SSL requests. Retrieves the value of a header in the reply resulting from a DO_HTTP command.

[DO_GetLastHttpError](#)

Applies to HTTP and SSL requests. Retrieves the integer indicating the error code of the last HTTP request sent with DO_Http. Errors greater than 399 include the Page not found 404 error.

[DO_GetRedirectedURL](#)

Applies to HTTP requests. Modifies the parameter passed in for use in the next request.

[DO_GetReplyBuffer](#)

Applies to HTTP and SSL requests. DO_GetReplyBuffer returns the HTTP response from the last DO_Http request.

[DO_GetUniqueString](#)

Applies to HTTP and SSL requests. Used to parse the most recent HTTP server reply to get the contents of a string that occurs between the left and right input strings.

[DO_GetUniqueStringEx](#)

Applies to HTTP and SSL requests. Used to parse a null-terminated input string (search) to get the contents of a string that occurs between the left and right input strings.

[DO_Http](#)

Applies to HTTP requests. Executes an HTTP request in the script.

[DO_HttpCleanup](#)

Applies to HTTP and SSL requests. Performs all necessary cleanup operations when a script exits or the user terminates the script.

[DO_HttpVersion](#)

Applies to HTTP and SSL requests. Specifies the version to use in the requests sent during playback.

[DO_InitHttp](#)

Applies to HTTP and SSL requests. Also applies to Visual Scripting. Sets all necessary internal variables needed to load test an HTTP script.

[DO_IPSpooferEnable](#)

Applies to HTTP and SSL requests. Enables each virtual user to appear to the web server as being sourced from a different network interface card.

[DO_NTLMAuthorization](#)

Applies to HTTP requests. Provides user ID and password (plain text or encrypted) information for NTLM authentication.

[DO_ProxyAuthorization](#)

Provides the username and password to access a password protected proxy server.

[DO_ProxyExceptions](#)

Applies to HTTP and SSL requests. Tells QALoad not to use the proxy server for hosts in the proxy exceptions list, so you can replay requests both inside and outside of the firewall in the same script.

[DO_SaveReplyType](#)

Applies to HTTP and SSL requests. Specifies types of replies to save.

[DO_SetAssumedContentType](#)

Applies to HTTP and SSL requests. Sets the default content type if the web server doesn't send a content type header.

[DO_SetBaudRate](#)

Returns the baud rate the virtual user will use.

[DO_SetBaudRateEx](#)

Returns the transmission rate the virtual user will use.

[DO_SetCheckpointName](#)

Sets the name of the next automatic checkpoint for the next DO_Http or DO_Https statement in the script.

[DO_SetCookie](#)

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

[DO_SetCookieEx](#)

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

[DO_SetJavascriptCleanupThreshold](#)

Applies to HTTP and SSL requests. Periodically QALoad will destroy its internal JavaScript model and recreate it. `DO_SetJavascriptCleanupThreshold` sets a count of the number of times JavaScript parsing is done before destroying and recreating the model.

[DO_SetMaxBrowserThreads](#)

Applies to HTTP and SSL requests. Specifies the number of concurrent connections to make for playback.

[DO_SetMaximumRetries](#)

Similar to the behavior of Netscape and Internet Explorer.

[DO_SetRefreshTimeout](#)

Specifies how long to wait for a meta refresh.

[DO_SetRetryWait](#)

Applies to SSL requests. Sets the proxy authorization when accessing SSL pages passed through a proxy server (also known as SSL tunneling).

[DO_SetTimeout](#)

Applies to HTTP and SSL requests. Specifies how long to wait for a reply from the server. If a reply is not received within the specified time, the virtual user will fail with a fatal error.

[DO_UseEntityList](#)

Applies to HTTP and SSL requests. Decodes non-ASCII character entities.

[DO_UseNumericReferenceList](#)

Applies to HTTP and SSL requests. Decodes non-ASCII numeric references.

[DO_UsePersistentConnections](#)

Applies to HTTP and SSL requests. Turns the use of persistent connections on or off.

[DO_UseProxy](#)

Applies to HTTP and SSL requests. Specifies a proxy server to use during testing.

[DO_UseProxyAutomaticConfiguration](#)

Applies to HTTP and SSL requests. Downloads the proxy automatic configuration (PAC) script at the specified URL. The rest of the transaction will use the PAC script to determine which proxy, if any, to connect to hosts.

[DO_VerifyDocTitle](#)

Applies to HTTP and SSL requests. Compares the parameters and match type passed in the parameters against the HTML page title specified in the response received from the HTTP request.

[DownloadMediaFromASX](#)

Applies to Windows Media Player Streaming Media. Dynamically parses an ASX file from the previous response and initiates and waits for completion of the specified Windows Media resources download.

[DownloadMediaRP](#)

Applies to Real Networks Streaming Media. Initiates and waits for completion of the specified multi-media resource download.

[DownloadWindowsMedia](#)

Applies to Windows Media Player Streaming Media. Initiates and waits for completion of the specified Windows Media resource download.

[EnableStatisticsRP](#)

Applies to Real Networks Streaming Media. Enables capture of media player performance statistics during a load test. Compuware recommends that this function is called in the initial section of a Web script, before the SYNCHRONIZE() call. Although it can be called at any point in the script, this command must appear in the script prior to any DownloadMediaRP call.

Fill_In

Applies to Visual Scripting. Used to represent how the user filled in fields on a form before clicking on a submit button. The values that are passed to Fill_In are expected to be plain text with no encoding other than using + to join multiple selects for LIST_BOX.

Get

Applies to Visual Scripting. Retrieves data from the virtual browser. Whole pages, specific frames, and text strings from within the document can be retrieved.

Navigate_To

Applies to Visual Scripting. Reads a URL typed in the Web browser's address field and constructs a request to navigate to the URL, or reads another request typed in the browser's address field, finishes the request and navigates to the request. Navigate_To is a direct replacement for DO_Http.

PlayMedia

Applies to Real Networks and Windows streaming media. Initiates and plays back the streaming media file that was stored in a previous call to the Click_On function.

Post_To

Applies to Visual Scripting. Reads a URL typed in the Web browser's address field as well as the encoding type. It then constructs a request to send a post to the URL.

RandNumString

Applies to Visual Scripting. Generates a random number from minimum to maximum.

Region

Applies to Visual Scripting. Marks the region_number parameter as an image map region.

RESTART_TRANSACTION_BOTTOM

Applies to Visual Scripting. Used to define a point at the end of the transaction for anything that needs to be deallocated or uninitialized. When transaction restarting occurs for a failed transaction, QALoad will first execute any code starting after the call to RESTART_TRANSACTION_BOTTOM allowing you to clean up important information and prevent memory leaks before retrying the transaction.

RESTART_TRANSACTION_TOP

Used to define a point at the beginning of the transaction loop that QALoad can use to rewind the transaction if the transaction fails and Restart Transaction error handling has been selected in the QALoad Conductor as follows:

Set

Applies to Visual Scripting. Assigns values to the Virtual Browser, Proxy, and other parts of the QALoad replay. This command sets the properties and attributes of the script.

ShowMediaRP

Applies to Real Networks Streaming Media. Displays the media during a load test. Audio and video can be controlled separately. If video is enabled, a dialog box displays the video. For audio, the sound from the media will play through the sound device.

Verify

Applies to Visual Scripting. Used to verify expected text against an element of the page just requested.

WWW_FATAL_ERROR

Applies to HTTP and SSL requests. Also applies to Visual Scripting. WWW_FATAL_ERROR aborts or restarts a virtual user in the event of an error during replay.

X_Coord

Applies to Visual Scripting. Marks the x_value parameter as an x-coordinate value.

XMLRequest

Applies to Visual Scripting. The XMLRequest function takes in the HTTP action and a URL and constructs a request to navigate to the URL. XMLRequest is a direct replacement for Navigate_To when the main HTTP request is for an XML document.

Y_Coord

Applies to Visual Scripting. Marks the y_value parameter as a y-coordinate value.

HTML character entities and numeric references

Applies to HTTP and SSL requests. In HTML programming, reserved characters (<, >, &, and ") and non-ASCII characters, such as the copyright and trademark symbol, are expressed in character entities or numeric references. A character entity contains an ampersand (&), followed by the entity name, followed by a semi-colon (;). For example, the character entity for less-than (<) is "<". A numeric reference contains an ampersand and pound sign, followed by the Unicode character code and a semi-colon. For example, the numeric reference for less-than (<) is "<".


For additional information on character encoding of low ASCII data (characters below ASCII 32) in QALoad, see [QALoad character encoding](#).

HTML Entities

QALoad automatically decodes the standard ASCII character entities (<, >, &, and "), but does not decode non-ASCII character entities. If you have a script containing non-ASCII character entities, define an ENTITY_LIST block which will allow QALoad to successfully decode the non-ASCII character entities. An entity list contains the following syntax:

```
ENTITY_LIST ( [Entity List Name] )
{
    ENTITY ( [Entity Name], [Character] )
    ENTITY ( [Entity Name], [Character] )
    ...
}
```

"Entity List Name" is the name given to the entity list. "Entity Name" is a string containing an HTML entity name. "Character" is a string containing the character to substitute in place of the entity.

 **Note:** An ENTITY_LIST must be defined outside of rrobot_script.

An entity list follows:

```
ENTITY_LIST ( myEntities )
{
    ENTITY ( "reg", "@" )
    ENTITY ( "copy", "©" )
    ENTITY ( "deg", "°" )
}
```

ENTITY_LIST creates a table that QALoad uses. For ENTITY_LIST to work within a script, place the command DO_UseEntityList() inside of rrobot_script. To use an entity list:

```
int rrobot_script ( PLAYERINFO* s_info )
{
```

```

    ...
    DO_UseEntityList(myEntities);
    ...
}

```

HTML Numeric References

QALoad automatically decodes all ASCII numeric references ("�" - ""), but does not decode non-ASCII numeric references. If you have a script containing non-ASCII numeric references, define a NUMERIC_REFERENCE_LIST block which will allow QALoad to successfully decode the non-ASCII numeric references. An entity list contains the following syntax:

```

NUMERIC_REFERENCE_LIST ( [Numeric Reference List Name] )
{
    NUMERIC_REFERENCE ( [Unicode Number], [Character] )
    NUMERIC_REFERENCE ( [Unicode Number], [Character] )
    ...
}

```

"Numeric Reference List Name" is the name given to the entity list. "Unicode Number" is the Unicode number of the character. "Character" is a string containing the character to substitute in place of the reference.

 **Note:** Define a NUMERIC_REFERENCE_LIST outside of rrobot_script as follows:

```

NUMERIC_REFERENCE_LIST ( myReferences )
{
    NUMERIC_REFERENCE ( 174, " " )
    NUMERIC_REFERENCE ( 169, " " )
    NUMERIC_REFERENCE ( 176, " " )
}

```

NUMERIC_REFERENCE_LIST creates a table that QALoad uses. To use NUMERIC_REFERENCE_LIST in a script, place DO_UseNumericReferenceList() inside of rrobot_script as follows:

```

int rrobot_script ( PLAYERINFO* s_info )
{
    ...
    DO_UseNumericReferenceList(myReferences);
    ...
}

```

Attach

Applies to Visual Scripting. Changes the current page to the page or frame specified.

This new page becomes the active page that all script commands act on.

Syntax

```
boolean Attach ( page_id );
```

Return Value

True if the page was attached to.

False if not.

Parameters

Parameter	Description
page_id	The page to attach to.

Example

```
page = Get ( PAGE );
Attach ( page );
Attach ( Get ( FRAME, "index" ) );
```

Clear

Applies to Visual Scripting. Used to clear out certain items such as the cache or cookies. Types can be ordered together to clear both.

Prototypes

```
boolean Clear ( enumeration type );
```

Return Value

True if cleared successfully.
False if not cleared successfully .

Parameters

Parameter	Description																								
type	<p>The type of clear to do. Valid types are listed in the following table.</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ALL</td> <td>Clear all internal settings.</td> </tr> <tr> <td>ALL_CGI_PARAMETERS</td> <td>Clear all Set CGI_PARAMETER parameters.</td> </tr> <tr> <td>ALL_COOKIES</td> <td>Clear all stored cookies. (Transaction type)</td> </tr> <tr> <td>ALL_HEADERS</td> <td>Clear all Set HEADER header attributes.</td> </tr> <tr> <td>ALL_VALUES</td> <td>Clear all DO_SetValue variables. (Transaction type)</td> </tr> <tr> <td>ATTACHED_FILES</td> <td>Clear all files in the binary file list.</td> </tr> <tr> <td>BASIC_AUTH_FLAG</td> <td>Do not send the basic authorization until next challenge. (Transaction type)</td> </tr> <tr> <td>BASIC_AUTHORIZATION</td> <td>Clear the basic authorization user name and password.</td> </tr> <tr> <td>BAUD_RATE_CALCULATIONS</td> <td>Clear the accumulated modern emulation data. (Transaction type)</td> </tr> <tr> <td>BLOCK_TRAFFIC_FROM</td> <td>Clear the blocked traffic from list.</td> </tr> <tr> <td>CACHE</td> <td>Clear out any virtual browser cache. (Transaction type)</td> </tr> </tbody> </table>	Type	Description	ALL	Clear all internal settings.	ALL_CGI_PARAMETERS	Clear all Set CGI_PARAMETER parameters.	ALL_COOKIES	Clear all stored cookies. (Transaction type)	ALL_HEADERS	Clear all Set HEADER header attributes.	ALL_VALUES	Clear all DO_SetValue variables. (Transaction type)	ATTACHED_FILES	Clear all files in the binary file list.	BASIC_AUTH_FLAG	Do not send the basic authorization until next challenge. (Transaction type)	BASIC_AUTHORIZATION	Clear the basic authorization user name and password.	BAUD_RATE_CALCULATIONS	Clear the accumulated modern emulation data. (Transaction type)	BLOCK_TRAFFIC_FROM	Clear the blocked traffic from list.	CACHE	Clear out any virtual browser cache. (Transaction type)
Type	Description																								
ALL	Clear all internal settings.																								
ALL_CGI_PARAMETERS	Clear all Set CGI_PARAMETER parameters.																								
ALL_COOKIES	Clear all stored cookies. (Transaction type)																								
ALL_HEADERS	Clear all Set HEADER header attributes.																								
ALL_VALUES	Clear all DO_SetValue variables. (Transaction type)																								
ATTACHED_FILES	Clear all files in the binary file list.																								
BASIC_AUTH_FLAG	Do not send the basic authorization until next challenge. (Transaction type)																								
BASIC_AUTHORIZATION	Clear the basic authorization user name and password.																								
BAUD_RATE_CALCULATIONS	Clear the accumulated modern emulation data. (Transaction type)																								
BLOCK_TRAFFIC_FROM	Clear the blocked traffic from list.																								
CACHE	Clear out any virtual browser cache. (Transaction type)																								

CERTIFICATE	Clear the client certificate.
CERTIFICATE_PASSWORD	Clear the client certificate password.
CONNECTION	Reset network connection. (Transaction type)
CONNECT_REQUEST_FOR_SSL_TUNNELING	Clear the set SSL connect string.
DEFAULT_CONTENT_TYPE	Clear the default content type.
DNS_CACHE	Clear any cached DNS lookups. (Transaction type)
JAVASCRIPT_ENGINE	Clear the Javascript state.
NTLM_AUTHORIZATION	Clear the NTLM user name and password.
ONLY_ALLOW_TRAFFIC_FROM	Clear the allowed traffic from list.
ONLY_USE_SSL_CIPHER	Clear the only use SSL cipher string.
PROXY_AUTHORIZATION	Clear the proxy authorization user name and password.
PROXY_AUTH_FLAG	Do not send the basic authorization until next challenge. (Transaction type)
PROXY_SETTINGS	Clear all proxy settings.
RECEPTION_BAUD_RATE	Turn off reception baud rate emulation.
REFERER	Clear the referer so it is not sent with the next request. (Transaction type)
SIGNIFICANT_CONTENT_TYPES	Clear the significant content type list.
SPOOFED_IP_ADDRESS	Clear any spoofed IP addresses.
TRANSACTION	Clear all temporary variables used in the transaction.
TRANSMISSION_BAUD_RATE	Turn off transmission baud rate emulation.

Examples

```
Clear ( JAVASCRIPT_ENGINE );
Clear ( CACHE );
Clear ( CONNECTION );
Clear ( ALL_COOKIES );
```

```
Clear ( TRANSACTION );
Clear ( ALL );
```

Click_On

Applies to Visual Scripting. Mimics the user clicking on text links, clickable images, and submit buttons.

Prototypes

```
boolean Click_On ( enumeration link, string description );
boolean Click_On ( enumeration link, integer count );
boolean Click_On ( enumeration link, enumeration specifier, string description );
boolean Click_On ( enumeration link, integer count, enumeration specifier, string
description );
boolean Click_On ( enumeration link, integer count, enumeration specifier, string
description, string x_coord, string y_coord );
boolean Click_On ( enumeration link, integer count, enumeration specifier, string
description, enumeration click_option );
boolean Click_On ( enumeration link, integer count, enumeration specifier, string
description, string region );
```

Return Value

Returns true if the requested link is found and the page is successfully retrieved. Otherwise, returns false.

Parameters

Parameter	Description								
link	<p>The type of link to try to click on. Valid types include:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>BUTTON</td> <td>A button.</td> </tr> <tr> <td>IMAGE</td> <td>A hyperlinked image.</td> </tr> <tr> <td>LINK</td> <td>A hyperlinked text anchor.</td> </tr> </tbody> </table>	Type	Description	BUTTON	A button.	IMAGE	A hyperlinked image.	LINK	A hyperlinked text anchor.
Type	Description								
BUTTON	A button.								
IMAGE	A hyperlinked image.								
LINK	A hyperlinked text anchor.								
description	The text of the link, or the appropriate text expected by a specifier. Can be used to specify URLs to use in conjunction with streaming media calls.								

<p>specifier</p>	<p>The way the text is used to find the link. Types of specifiers are listed in the following table:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>NAME_ATTRIBUTE</td> <td>Name attribute of the HTML tag.</td> </tr> <tr> <td>SRC_ATTRIBUTE</td> <td>Src attribute of the HTML tag.</td> </tr> <tr> <td>ALT_ATTRIBUTE</td> <td>Alt attribute of the HTML tag.</td> </tr> <tr> <td>DESCRIPTION</td> <td>Description as seen by the user in the browser (default).</td> </tr> <tr> <td>CONTAINING</td> <td>Click_On link that contains unique HTML code in the page.</td> </tr> <tr> <td>BEFORE</td> <td>Click_On link before unique HTML code in the page.</td> </tr> <tr> <td>AFTER</td> <td>Click_On link after unique HTML code in the page.</td> </tr> </tbody> </table>	Type	Description	NAME_ATTRIBUTE	Name attribute of the HTML tag.	SRC_ATTRIBUTE	Src attribute of the HTML tag.	ALT_ATTRIBUTE	Alt attribute of the HTML tag.	DESCRIPTION	Description as seen by the user in the browser (default).	CONTAINING	Click_On link that contains unique HTML code in the page.	BEFORE	Click_On link before unique HTML code in the page.	AFTER	Click_On link after unique HTML code in the page.
Type	Description																
NAME_ATTRIBUTE	Name attribute of the HTML tag.																
SRC_ATTRIBUTE	Src attribute of the HTML tag.																
ALT_ATTRIBUTE	Alt attribute of the HTML tag.																
DESCRIPTION	Description as seen by the user in the browser (default).																
CONTAINING	Click_On link that contains unique HTML code in the page.																
BEFORE	Click_On link before unique HTML code in the page.																
AFTER	Click_On link after unique HTML code in the page.																
<p>count</p>	<p>The nth match. For example, if a Web page has three buttons with the same description, such as Submit, use a count of 3 to match the third button.</p>																
<p>x_coord</p>	<p>The X coordinate of the server-side image map.</p>																
<p>y_coord</p>	<p>The Y coordinate of the server-side image map.</p>																
<p>region</p>	<p>The region of a client-side image map to click on.</p>																
<p>click_option</p>	<p>The encoding format for any CGI parameters of the URL. Valid values are listed in the following table:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>CLICK_BUTTON</td> <td>Click on a button to submit a form. (Default)</td> </tr> <tr> <td>PRESS_ENTER_KEY</td> <td>Press the Enter key on a control to submit a form.</td> </tr> </tbody> </table>	Value	Description	CLICK_BUTTON	Click on a button to submit a form. (Default)	PRESS_ENTER_KEY	Press the Enter key on a control to submit a form.										
Value	Description																
CLICK_BUTTON	Click on a button to submit a form. (Default)																
PRESS_ENTER_KEY	Press the Enter key on a control to submit a form.																

Examples

```
Click_On ( LINK, "Click Here" );
Click_On ( IMAGE, SRC_ATTRIBUTE, "image.gif" );
Click_On ( BUTTON, 2, CONTAINING, "<button name=' ' " );
Click_On ( LINK, 4 );
Click_On ( BUTTON, 1, DESCRIPTION, "Submit Query", PRESS_ENTER_KEY);
```


Additional Notes on How Click_On Handles Image Maps

There are two types of image maps: client side and server side. Client side image maps expect special handling by the client so that when a user clicks an area (region) of the image map, a URL is requested that is particular to that region. Server side image maps add on the X and Y coordinates of where the user clicked in the image map. Client side image maps pass one parameter to Click_On which represents the region. Server side image maps will pass two parameters to Click_On which represent the X and Y coordinates where the image was clicked.

The region parameter of a client side image map will be a count of which region (area or a tag) to be used for the Click_On. The count will be in order of how they are listed in the HTML. A Region macro is defined to give the additional parameter a better label, so it is easily recognized as the region parameter. If a client side image map is detected and no region parameter is passed, Click_On will automatically select the first region.

Server side image maps coordinate parameters which are a representation of where on the image map the user wants to click. X_Coord and Y_Coord macros are defined so they are easily recognized as coordinate parameters. If a server side image map is detected and no coordinates are passed, Click_On will automatically supply an X coordinate of 0 and a Y coordinate of 0.

Examples

```
Click_On ( IMAGE, "Click Here", Region (1) );
Click_On ( IMAGE, 2, AFTER, "Fun Fun", Region (1) );
Click_On ( IMAGE, 4, Region (2) );
Click_On ( IMAGE, "Click Here", X_Coord (10), Y_Coord (11) );
Click_On ( IMAGE, 2, AFTER, "Fun Fun", X_Coord (20), Y_Coord (30) );
Click_On ( IMAGE, 4, X_Coord (20), Y_Coord (60) );
```

DisableStatisticsRP

Applies to Real Networks Streaming Media. Disables capture of statistics during a load test.

Notes:

- ! Real Player streaming media is only supported in process mode. On the QALoad Player main window, in the Run As: group, select the Process option.
- ! For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad " chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

```
DisableStatisticsRP();
```

Parameters

None.

Example


```
DisableStatisticsRP();
```

DO_AddHeader

Applies to HTTP and SSL requests. Indicates headers that are common to every request (DO_Http or DO_Https function calls) in a script.

QALoad takes all of the headers that are in every request in the script and places them at the beginning of the script (between BEGIN_TRANSACTION and the first request) using the DO_AddHeader command.

During replay, DO_AddHeader tells QALoad to add the header with a given name and value to all requests in the script.

 **Note:** Cookie and Host headers are not included in the DO_AddHeader function even if they are common to all of the requests.

Syntax

DO_AddHeader (const char *name, const char *value);

Parameters

Parameter	Description
Name	The name of the header.
Value	The value of the header.

Example

If the following two requests occurred in the same script, the User-Agent header would be considered common:

```
DO_Http("GET http://yourserver.net/ HTTP/1.0\r\n"
"User-Agent: Mozilla/4.7 [en] (WinNT; I)\r\n"
"Host: yourserver.net\r\n"
"Accept: */*\r\n"
"Accept-Language: ja_JP\r\n"
"Accept-Charset: *\r\n\r\n");
DO_Http("GET http://anotherserver.net/ HTTP/1.0\r\n"
"User-Agent: Mozilla/4.7 [en] (WinNT; I)\r\n"
"Host: anotherserver.net\r\n"
"Accept: image/jpeg, */*\r\n"
"Accept-Language: en\r\n"
"Accept-Charset: iso-8859-1,*,utf-8\r\n\r\n");
```

Note that both User-Agent and Mozilla/4.7 [en] (WinNT; I) must be the same in order for them to be considered common. Using the above example, the resulting script will look like this:

```
DO_AddHeader("User-Agent", "Mozilla/4.7 [en] (WinNT; I)");
DO_Http("GET http://yourserver.net/ HTTP/1.0\r\n"
"Host: yourserver.net\r\n"
"Accept: */*\r\n"
"Accept-Language: ja_JP\r\n"
"Accept-Charset: *\r\n\r\n");
DO_Http("GET http://anotherserver.net/ HTTP/1.0\r\n"
"Host: anotherserver.net\r\n"
"Accept: image/jpeg, */*\r\n"
"Accept-Language: en\r\n"
"Accept-Charset: iso-8859-1,*,utf-8\r\n\r\n");
```

Note that the User-Agent header was removed from the DO_Http() calls. The DO_AddHeader() call tells QALoad to add the header with a given name and value to all requests in the script.

DO_AllowTrafficFrom

Applies to HTTP and SSL requests. If DO_AllowTrafficFrom is present in a script, then sub-requested URLs only occur if the sub-request's URL contains one of the sub-strings in the 'substrings' list.

For example, if substrings is "www.host.com, images; .js", then the following URLs could be sub-requested.

```
http://www.host.com/top-frame.html : URL has a substring "www.host.com"
http://img.host.com/images/fist.png : URL has a sub-string "images"
http://scripts.host.com/scripts/menu.js : URL has a sub-string ".js"
```

And the following URL could not be sub-requested:

`http://x.host.com/no-reason-to-request/page.html` : No substring found

Syntax

```
DO-AllowTrafficFrom ( const char * substrings )
```

Parameters

Parameter	Description
substrings	Semicolon separated list of sub-strings.

Example

```
DO-AllowTrafficFrom ( "www.host.com; images; .js" );
```

DO_AttachFile

Applies to HTTP and SSL requests. Specifies files that should be loaded into memory at the beginning of a script run. This is used in Post transactions that include binary files.

Syntax

```
DO_AttachFile(const char *label, const char *filename);
```

Return Value

Parameters

Parameter	Description
label	The variable that is replaced by the file contents in the request at run-time.
filename	The relative filename for the file to attach.

Example

```
...
...
DO_AttachFile("FILE_1", "mee-1.jpg");
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
"Content-Disposition: frm-data; name=\"phylenam\"; filename="
 "\"F:\\temp\\mee-1.jpg\""\r\n"
"Content-Type: image/pjpeg\r\n\r\n{*FILE_1}\r\n"
"-----7d02d1b240910--");
...
...
```

DO_AutomaticSubRequests

Applies to HTTP requests. Indicates whether subrequests will be downloaded during replay.

This command relates to the Automatically Process HTTP SubRequests check box on the QALoad Script Development Workbench Convert Options wizard. When this option is selected,

DO_AutomaticSubRequests (TRUE); is written to the script when it is converted from a capture file and subrequests are not included in the script. During replay, QALoad handles subrequests like a browser.

When it is not selected, DO_AutomaticSubRequests(FALSE); is written to the script when it is converted from a capture file. Each DO_Http request evaluates the Web page, determines if it contains any subrequests (requests that call for images, style sheets, or XML DTD's), and downloads these items. All subrequests in the capture file are converted into the resulting script and executed during replay.

By default, the Automatically Process HTTP SubRequests check box is selected. DO_AutomaticSubRequests is placed at the beginning of a script, between the BEGIN_TRANSACTION command and the first request.

Syntax

```
int DO_AutomaticSubRequests (BOOL bFlag);
```

Return Value

0 if bFlag is set to FALSE.

1 if bFlag is set to TRUE.


Parameters

Parameter	Description
bflag	A flag indicating if the Automatically Process HTTP SubRequests option is enabled (TRUE or FALSE).

Examples

Example 1:

The following example is valid when bflag is TRUE:

 **Note:** Request 3 (logo.gif) is not present when SubRequest is TRUE.

```
...
...
BEGIN_TRANSACTION();
...
...
DO_AutomaticSubRequests(TRUE);
...
...
/* Request: 1 */
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
"Accept: image/gif, image/x-xbitmap, */*\r\n"
"Host: company.com\r\n"
"Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
...
...
/* Request: 2 */
DO_Http("GET http://company.com/index.htm HTTP/1.0\r\n"
"Accept: image/gif, image/x-xbitmap, */*\r\n"
"Referer: http://company.com/\r\n"
"Host: company.com\r\n"
"Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
...
...
END_TRANSACTION();
...
...
```

Example 2:

The following example is valid when `bflag` is FALSE:

```

...
...
BEGIN_TRANSACTION();
...
...
DO_AutomaticSubRequests(FALSE);
...
...
/* Request: 1 */
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
"Accept: image/gif, image/x-xbitmap, */*\r\n"
"Host: company.com\r\n"
"Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
...
...
/* Request: 2 */
DO_Http("GET http://company.com/index.htm HTTP/1.0\r\n"
"Accept: image/gif, image/x-xbitmap, */*\r\n"
"Referer: http://company.com/\r\n"
"Host: company.com\r\n"
"Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
...
...
/* Request: 3 */
DO_Http("GET http://company.com/logo.gif HTTP/1.0\r\n"
"Accept: */*\r\n"
"Referer:http://company.com/index.htm\r\n"
"Host: company.com\r\n"
"Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
...
...
END_TRANSACTION();
...
...
/

```

DO_BasicAuthorization

Specifies the username and password to gain access to a password protected WWW host, directory, or file.

The password may be encrypted using QALoad's "~encr~" encryption. Username and password are inserted automatically as necessary during conversion. Note that you can variablize the username and password to emulate different users accessing the resources.

Syntax

```
DO_BasicAuthorization(const char *username, const char *password);
```

Parameters

Parameter	Description
username	A valid username for the resource you're attempting to access.
password	The associated password.

Example


```
DO_HttpVersion("Auto");
DO_SLEEP(2);

/* Request: 1 */

DO_BasicAuthorization("smith", "~encr~0E636502080E");
BeginCheckpoint(" http://iris/redline - chkpt: 1");
DO_Http("GET http://iris/redline HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, "
"application/vnd.ms-excel, application/msword, "
"application/vnd.ms-powerpoint, */*\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT; CPWR)\r\n"
"Host: iris\r\n\r\n"
);
```

DO_BlankOutOfRangeData

Applies to HTTP and SSL requests. If DO_BlankOutOfRangeData is enabled, then characters in the HTTP response body that interface with text searching or the HTML parser are changed to spaces.

 **Note:** Currently, the only character blanked by DO_BlankOutOfRangeData is the NUL character (ASCII value 0).

Syntax

```
DO_BlankOutOfRangeData ( BOOL flag );
```

Parameters

Parameter	Description
flag	Flag indicating if out of range blanking is to be done or not.

Example

```
DO_BlankOutOfRangeData ( TRUE );
```

DO_BlockTrafficFrom

Applies to HTTP and SSL requests. If DO_BlockTrafficFrom is present in a script, then sub-requested URLs only occur if the sub-request's URL does not contain one of the sub-strings in the 'substrings' list.

For example, if sub-strings list is "pop-up; imgsrv", then the following URLs would not be sub-requested:

```
http://www.host.com/pop-up/ad.html : URL has a substring "pop-up"
http://imgsrv.host.com/images/fist.png : URL has a sub-string "imgsrv"
```

And the following URL could be sub-requested:

```
http://www.host.com/no-reason-to-block/page.html : No substring found
```

Syntax

```
DO_BlockTrafficFrom( const char * substrings )
```

Parameters

Parameter	Description
substrings	Semicolon separated list of sub-strings.

Example

```
DO_BlockTrafficFrom ( "pop-up; imgsrv" );
```

DO_Cache

Applies to HTTP and SSL requests. Turns on cache emulation, which caches anything with a content type beginning with "image/".

DO_Cache is related to the Cache option on the WWW Advanced options dialog box. If that option is selected, DO_Cache (TRUE); is written into the script during the convert process, and requested images are cached.

Syntax

```
DO_Cache(BOOL flag);
```

Parameters

Parameter	Description
flag	TRUE (on) or FALSE (off).

Example

```
DO_InitHttp(s_info);
DO_Cache(); /* Enable cache */
```

DO_Clear

Applies to HTTP and SSL requests. Used to clear out certain items, such as the cache or cookies. Types can be ordered together to clear both.

Prototypes

```
DO_Clear ( enumeration type );
```

Parameters

Parameter	Description												
type	<p>The type of clear to do. Valid types are listed in the following tables:</p> <p>Values for Meta type</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>TRANSACTION</td> <td>Clear all temporary variables used in transaction.</td> </tr> <tr> <td>ALL</td> <td>Clear everything.</td> </tr> </tbody> </table> <p>Values for Transaction type</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ALL_COOKIES</td> <td>Clear all stored cookies.</td> </tr> <tr> <td>BASIC_AUTH_FLAG</td> <td>Do not send the basic authorization until next</td> </tr> </tbody> </table>	Value	Description	TRANSACTION	Clear all temporary variables used in transaction.	ALL	Clear everything.	Value	Description	ALL_COOKIES	Clear all stored cookies.	BASIC_AUTH_FLAG	Do not send the basic authorization until next
Value	Description												
TRANSACTION	Clear all temporary variables used in transaction.												
ALL	Clear everything.												
Value	Description												
ALL_COOKIES	Clear all stored cookies.												
BASIC_AUTH_FLAG	Do not send the basic authorization until next												

	challenge.
BAUD_RATE_CALCULATIONS	Clear the accumulated modem emulation data.
CACHE	Clear out any virtual browser cache.
CONNECTION	Reset network connection.
DNS_CACHE	Clear any cached DNS lookups.
PROXY_AUTH_FLAG	Do not send the basic authorization until next challenge.
REFERER	Clear the referer so it is not sent with the next request.

Values for ALL type

Value	Description
ALL_CGI_PARAMETERS	Clear all Set CGI_PARAMETER parameters.
ALL_HEADERS	Clear all Set HEADER header attributes.
ATTACHED_FILES	Clear all files in the binary file list.
BASIC_AUTHORIZATION	Clear the basic authorization username and password.
BLOCK_TRAFFIC_FROM	Clear the block traffic from list.
CERTIFICATE	Clear the client certificate.
CONNECT_REQUEST_FOR_SSL_TUNNELING	Clear the set SSL connect string.
DEFAULT_CONTENT_TYPE	Clear the default content type.
JAVASCRIPT_ENGINE	Clear the JavaScript state.
NTLM_AUTHORIZATION	Clear the NTLM username, password, and domain.
ONLY_ALLOW_TRAFFIC_FROM	Clear the allow traffic from list.
ONLY_USE_SSL_CIPHER	Clear the only use SSL cipher string.
PROXY_AUTHORIZATION	Clear the proxy authorization username and password.
PROXY_SETTINGS	Clear all proxy settings.


	RECEPTION_BAUD_RATE	Turn off reception baud rate emulation.
	SIGNIFICANT_CONTENT_TYPES	Clear the significant content type list.
	SPOOFED_IP_ADDRESS	Clear any spoofed IP address.
	TRANSMISSION_BAUD_RATE	Turn off transmission baud rate emulation.

Examples

```
DO_Clear ( JAVASCRIPT_ENGINE );
DO_Clear ( CACHE );
DO_Clear ( CONNECTION );
DO_Clear ( ALL_COOKIES );
DO_Clear ( TRANSACTION );
DO_Clear ( ALL );
```

DO_ClearCache

Applies to HTTP and SSL requests. Clears any cached images. Performed automatically by DO_HttpCleanup.

 **Note:** This command is the same as DO_Clear (CACHE).

Syntax

```
DO_ClearCache();
```

Parameters

None.

DO_ClearDNSCache

Applies to HTTP and SSL requests. When DO_Http or DO_Https make an HTTP request, DO_ClearDNSCache caches any DNS lookups that are performed. If that cache needs to be cleared to simulate browser, use DO_ClearDNSCache.

 **Note:** This command is the same as DO_Clear (DNS_CACHE).

Syntax

```
DO_ClearDNSCache()
```

Parameters

None.

Example

```
...
...
/* Request: 1 */
DO_Http ("GET http://company.com/ HTTP/1.0\r\n\r\n");
DO_ClearDNSCache();
/* Request: 2 */
```


QALoad 5.02

```
/*  
* Do a brand new DNS lookup of company.com  
*/  
DO_Http ("GET http://company.com/ HTTP/1.0\r\n\r\n" );  
...  
...
```

DO_ClearJavascript

Applies to HTTP and SSL requests. Clears any memory allocated by the JavaScript engine.

Performed automatically by DO_HttpCleanup. DO_ClearJavascript is the same as DO_Clear (JAVASCRIPT_ENGINE).

 **Note:** DO_ClearJavascript is a deprecated command. Compuware recommends that you use DO_SetJavascriptCleanupThreshold instead.

Syntax

```
DO_ClearJavascript();
```

Parameters

None.

Example

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

DO_DynamicCookieHandling

Applies to HTTP and SSL requests. Turns dynamic cookie handling on or off.

This command relates to the Enable Dynamic Cookie Handling option on the QALoad Script Development Workbench Convert Options wizard. When this option is selected, DO_DynamicCookieHandling (TRUE); is written to the script and the script does not include cookie-related statements (DO_GetCookieFromReplyEx, DO_SetValue, etc.).

When the Enable Dynamic Cookie Handling option is not selected, the converted script includes the statement DO_DynamicCookieHandling (FALSE); and includes all cookie-related information. By default, the Enable Dynamic Cookie Handling check box is selected.

Syntax

```
int DO_DynamicCookieHandling(BOOL bFlag)
```

Return Value

0 if bFlag is set to FALSE.

1 if bFlag is set to TRUE.

Parameters

Parameter	Description
bFlag	Indicates whether dynamic cookie handling is turned on (TRUE) or off FALSE.

Examples

Example 1:

When the Enable Dynamic Cookie Handling option is not selected:

```

...
...
/* Declare Variables */
char *Cookie[1];
...
...
for(i=0;i<1;i++)
Cookie[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_DynamicCookieHandling(FALSE);
...
...
DO_GetCookieFromReplyEx("NUM", &Cookie[0], '*');
/* Request: 4 */

DO_SetValue("cookie000", Cookie[0]);
DO_Http("GET http://company.com/cgi-bin/cookiespipes.pl "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Host: company.com\r\n"
        "Cookie: HTMLA=FONTSIZE=LARGE; {*cookie000}; "
        "SITESERVER=ID=4b4ab9751bce9a95f74ec62\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Cookie[i]);
Cookie[i]=NULL;
}
END_TRANSACTION();
...
...

```

Example 2:

When the Enable Dynamic Cookie Handling option is selected:

```

...
...
BEGIN_TRANSACTION();
...
...
DO_DynamicCookieHandling(TRUE);
...
...
DO_Http("GET http://company.com/cgi-bin/cookiespipes.pl "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Host: company.com\r\n"
        "Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID=" "4b4ab9751bce9a95f74ec62\r\n\r\n");
...
...
END_TRANSACTION();

```

...
...

DO_DynamicRedirectHandling

Applies to HTTP requests. Retrieves a redirected URL for use in the next request.

DO_DynamicRedirectHandling is related to the Enable Dynamic Redirect Handling option on the QALoad Script Development Workbench Convert Options wizard. If that option is selected, DO_DynamicRedirectHandling(TRUE) is written into the script during the convert process. The script then checks every response from a DO_HTTP for a 301, 302, 303, or 307 message and performs the redirected request.

This line should appear only once in the script and at the beginning of the script.

Syntax

```
int DO_DynamicRedirectHandling (BOOL bFlag)
```

Return Value

0 if bFlag is set to FALSE.

1 if bFlag is set to TRUE.

Parameters

Parameter	Description
bFlag	Flag that indicates whether redirection should be handled.

Examples

Example 1:

When Enable Dynamic Redirect Handling option is TRUE:

```
...
...
DO_DynamicRedirectHandling(TRUE);
...
...
/* Request: 4 To: Redirected Webpage */
DO_Http("GET http://examples.com/cgi-bin/dynredirect.exe "
        "HTTP/1.0\r\n"
        "Accept: image/gif, application/pdf, */*\r\n"
        "Referer: http://examples.com/index.htm\r\n"
        "Host: examples.com\r\n\r\n");
DO_Http("GET http://examples.com/nextrequest.htm"
        "HTTP/1.0\r\n"
        "Accept: image/gif, application/pdf, */*\r\n"
        "Referer: http://examples.com/redirect.htm\r\n"
        "Host: examples.com\r\n\r\n");
...

```

Example 2:

When Enable Dynamic Redirect Handling option is FALSE:

```
...
...
DO_DynamicRedirectHandling(FALSE);
...

```

```

...
/* Request: 4 To: Redirected Webpage */
DO_Http("GET http://examples.com/cgi-bin/dynredirect.exe "
        "HTTP/1.0\r\n"
        "Accept: image/gif, application/pdf, */*\r\n"
        "Referer: http://examples.com/index.htm\r\n"
        "Host: examples.com\r\n\r\n");

DO_Http("GET http://examples.com/redirect.htm"
        "HTTP/1.0\r\n"
        "Accept: image/gif, application/pdf, */*\r\n"
        "Referer: http://examples.com/cgi-bin/ dynredirect.exe\r\n"
        "Host: examples.com\r\n\r\n");

DO_Http("GET http://examples.com/nextrequest.htm"
        "HTTP/1.0\r\n"
        "Accept: image/gif, application/pdf, */*\r\n"
        "Referer: http://examples.com/redirect.htm\r\n"
        "Host: examples.com\r\n\r\n");
...

```

DO_EnableJavascript

Applies to HTTP requests. Enables or disables the interpretation of Javascript.

By default, QALoad attempts to interpret Javascript detected during replay. If you disable this feature, you may be able to reduce the amount of CPU overhead during WWW replay. However, this may cause WWW replay to miss some sub-requests and cookies contained in Javascript on HTML pages. To disable Javascript interpretation, insert `DO_EnableJavascript(FALSE);` into your script.

Syntax

```
DO_EnableJavascript(BOOL flag);
```

Return Value

True = on
False = off

Parameters

Parameter	Description
flag	Used to enable or disable the interpretation of Javascript.

Example

```

...
...
DO_EnableJavascript(FALSE);
BEGIN_TRANSACTION();
...
...

```

DO_EncodeString

Applies to HTTP and SSL requests. `DO_EncodeString` takes in a string and URL-encodes the string to be suitable to use as a CGI parameter or in the body of a POST.

Syntax

```
int DO_EncodeString( const char *szSource, char **pszDestination )
```

Return Value

The difference between the string length of the destination and the string length of the source.

Parameters

Parameter	Description
szSource	String to URL encode.
pszDestination	Address of a string (char*) to hold the URL encoded string.

Example

```
char* szEncoded= 0;
...
/* The value of szEncoded will be "a+string%21" */
DO_EncodeString( "a string!", &szEncoded );
```

DO_FreeHttp

Applies to HTTP and SSL requests. Also applies to Visual Scripting. Clears memory used by the script.

This command is used at the end of every HTTP script. DO_FreeHttp is automatically inserted during the convert process and should never need to be adjusted.

Syntax

```
DO_FreeHttp( );
```

Parameters

None.

Example

```
...
...
END_TRANSACTION();
DO_FreeHttp();
REPORT(SUCCESS);
```

DO_GetAnchorByNumber

Applies to HTTP and SSL requests. Stores the value of an anchor from an HTML reply into a string that can be substituted into subsequent requests.

This command is used when an anchor is embedded in an HTML reply at a known location, but the anchor text may change. For example, a search engine returns a page with 10 anchors in response to a query, and the business logic for the transaction requires clicking on the third anchor regardless of the text for that anchor.

Syntax

```
int DO_GetAnchorByNumber( int anchorNumber, char **anchorValue );
```

Returns

1 if successful
0 if unsuccessful

Parameters

Parameter	Description
anchorNumber	A number which is the count of the anchor to be retrieved.
anchorValue	Address to a string where the anchor value is stored.

Example

```

...
char *AnchorByNumber= NULL;
...
BEGIN_TRANSACTION();
...
DO_GetAnchorByNumber(3, &AnchorByNumber);
...
DO_SetValue("AnchorByNumber", AnchorByNumber);
...
DO_Http("GET { *AnchorByNumber } HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://company/cgi-bin/perl_9.pl\r\n"
        "Host: company\r\n"
        "Cookie: username=anu; c2_LastVisit=6\r\n\r\n");
...
if ( AnchorByNumber )
{
free(AnchorByNumber);
AnchorByNumber= NULL;
}
...
END_TRANSACTION();

```

DO_GetAnchorCount

Applies to HTTP and SSL requests. Returns the total number of the anchors on the page.

Syntax

```
int DO_GetAnchorCount()
```

Return Type

Integer

Parameters

none

Example

```

int n;
char *Anchor[1]= { NULL };
...
n= DO_GetAnchorCount();
DO_GetAnchorByNumber ( n/2, &Anchor[ 0 ] );

```

DO_GetAnchorHREF

Applies to HTTP and SSL requests. Stores the value of a named anchor off of an HTML reply into a string that can be substituted into subsequent requests.

This command is used when an anchor to a CGI request is embedded in a dynamic HTML reply (for instance, the results from a search engine query). The DO_GetAnchorHREF function is automatically inserted by QALoad during conversion whenever this situation is encountered.

Note: If you are adding commands, QALoad uses the following rules for matching the anchorName parameter to the tag and anchor text. To modify this command in a script, take the syntax in the attribute (the value for the alt= or src= tags) and append it to either the "alt=" or "src=" (case sensitive) attribute.

If there is an tag in the source HTML, use the alt= attribute.

Example:
 FOR: click
 USE: DO_GetAnchorHREF ("alt=look", Anchor [0]);

If the tag has no alt= attribute, use the src= attribute.

Example:
 FOR: click
 USE: DO_GetAnchorHREF ("src=look.gif", Anchor [0]);

If there is no tag, use the anchor text between <a> and .

Example:
 FOR: click here
 USE: DO_GetAnchorHREF ("click here", Anchor [0]);

The anchor text is made by removing all HTML tags and spaces. Words are extracted and put together separated by a single space.

Syntax

```
int DO_GetAnchorHREF( const char *anchorName, char **anchorValue );
```

Return Value

1 if successful.
 0 if unsuccessful.

Parameters

Parameter	Description
anchorName	String constant specifying the name of the anchor to retrieve from the reply.
anchorValue	Address to a string where the anchor value is stored.

Example

```
...
...
char *Anchor[1];
...
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetAnchorHREF( "Resubmit", &Anchor[0]);
DO_SetValue( "Anchor000", Anchor[0]);
DO_Http( "GET { *Anchor000} HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://company/cgi-bin/perl_9.pl\r\n"
        "Host: company\r\n"
```



```

        "Cookie: username=anu; c2_LastVisit=6\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
}
...
...
END_TRANSACTION();
...
...

```

DO_GetAnchorHREFEx

Applies to HTTP and SSL requests. Stores the value of a specific occurrence of a named anchor off an HTML reply into a string that can be substituted into subsequent requests.

This command is used when an anchor to a CGI request is embedded in a dynamic HTML reply, for example, the results from a search engine query, more than once. The which parameter specifies the occurrence of the anchor to retrieve. The DO_GetAnchorHREFEx function is automatically inserted by QALoad during conversion whenever this situation is encountered.

If you are adding commands to match the anchorName parameter to the tag and anchor text, see the note and examples for [DO_GetAnchorHREF](#).

Syntax

```
int DO_GetAnchorHREFEx( const char *anchorName, int count, char **anchorValue );
```

Return Value

- 1 if successful
- 0 if unsuccessful

Parameters

Parameter	Description
anchorName	String constant specifying the name of the anchor to retrieve from the reply.
count	Which occurrence of the anchor to retrieve.
anchorValue	Address to a string where the anchor value is stored.

Example

```

...
...
char *Anchor[1];
...
...
for(i=0;i<1;i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetAnchorHREFEx( "Resubmit", &Anchor[0], 1 );
DO_SetValue("Anchor000", Anchor[0]);
DO_Http("GET {*Anchor000} HTTP/1.0\r\n"
        "Accept: */*\r\n"

```

QALoad 5.02

```
"Referer: http://company/cgi-bin/perl_9.pl\r\n"
"Host: company\r\n"
"Cookie: username=anu; c2_LastVisit=6\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
}
...
...
END_TRANSACTION();
...
...
```

DO_GetAnchorHREFn

Applies to HTTP and SSL requests. Stores the value of a specific occurrence of a named anchor off of an HTML reply into a string that can be substituted into subsequent requests.

This command is used when an anchor to a CGI request is embedded in a dynamic HTML reply, for instance, the results from a search engine query, more than once. The which parameter specifies the occurrence of the anchor to retrieve. The DO_GetAnchorHREFn function is automatically inserted by QALoad during conversion whenever this situation is encountered.

If you are adding commands, to match the anchorName parameter to the tag and anchor text, see the note and examples under [DO_GetAnchorHREF](#).

Syntax

```
int DO_GetAnchorHREFn( const char *anchorName, char **anchorValue, int count );
```

Return Value

1 if successful
0 if unsuccessful

Parameters

Parameter	Description
anchorName	String constant specifying the name of the anchor to retrieve from the reply.
anchorValue	Address to a string where the anchor value is stored.
count	The occurrence of the anchor to retrieve.

Example

```
...
...
char *Anchor[1];
...
...
for(i=0; i<1; i++)
Anchor[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetAnchorHREFn( "Resubmit", &Anchor[0], 3);
DO_SetValue("Anchor000", Anchor[0]);
```

```
DO_Http("GET {*Anchor000} HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://company/cgi-bin/perl_9.pl\r\n"
        "Host: company\r\n"
        "Cookie: username=anu; c2_LastVisit=6\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Anchor[i]);
}
...
...
END_TRANSACTION();
...
...
```

DO_GetClientMapHREF

Applies to HTTP and SSL requests. DO_GetClientMapHREF is used to extract the href URL from a particular region of a client-side image map.

Client-side image maps are specified within an HTML document by the 'map' tag. Inside the 'map' tag, 'a' and 'area' tags are used to specify regions of the image map. The href attribute of the 'a' or 'area' tags specify the location of the URL to go to.

Syntax

```
BOOL DO_GetClientMapHREF ( int nMapCount, int nRegionCount, char ** pszURL );
```

Return Value

TRUE for successful
 FALSE for unsuccessful.

Parameters

Parameter	Description
sMapCount	Count of 'map' tags inside the HTML. The map count can be wrapped in the MAP macro to make the script more readable.
nRegionCount	Count of 'a' and 'area' tags inside of the 'map' tag. The region count can be wrapped in the REGION macro to make the script more readable.
pszURL	Address of a string pointer to hold the href URL for the map and region.

Example

```
char * ClientMapURL [1];
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http ("GET http://company.com/ HTTP/1.0\r\n\r\n" );
DO_GetClientMapHREF( MAP(1), REGION (1), &ClientMapURL [0] );
/* Request: 2 */
DO_SetValue ("ClientMap000", ClientMapURL [0] );
DO_Http ("GET {*ClientMap000} HTTP/1.1\r\n\r\n" );
```

...
...

DO_GetCookie

Applies to HTTP and SSL requests. Extracts a cookie from the QALoad internal cookie list.

The cookie is retrieved based on the name of the cookie. Wildcard patterns can be used to specify the cookie name in case the cookie name is dynamic. A count is also specified in case multiple cookies match the specified name.

 **Note:** DO_GetCookie requires DO_DynamicCookieHandling be set to TRUE.

Syntax

```
BOOL DO_GetCookie ( const char * szName, int nCount, char **
pszCookie );
```

Return Value

TRUE for successful
FALSE for unsuccessful

Parameters

Parameter	Description
szName	Name of the cookie to get. Wildcard patterns, like '*' to match anything can be used.
nCount	Count of which occurrence to get.
pszCookie	Address of a string pointer to hold the cookie.

Example

```
char * userid;
char * aspessionid;
...
...
BEGIN_TRANSACTION();
...
...

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

/*
 * Get a cookie named USER_ID
 */
DO_GetCookie ( "USER_ID", 1, &userid );

/*
 * Get the second ASPSESSIONID cookie. ASPSESSIONID
 * cookies always have extra characters on the end to make
 * them unique.
 *
 * An example ASPSESSIONID: ASPSESSIONIDQQGGQDO=EBOOONBBFH
 * BBELAJIMEFAKAP
 */
DO_GetCookie ( "ASPSESSIONID*", 2, &aspessionid );
```

DO_GetCookieFromReplyEx

Applies to HTTP and SSL requests. Retrieves and stores the value of a cookie when a Set-Cookie: statement is encountered in a reply header.

A stored cookie can be used later in the script in a DO_SetValue command to pass the cookie value on to subsequent requests. QALoad's Convert facility automatically inserts a DO_GetCookieFromReplyEx into the script if it detects a Set-Cookie: header field.

Although this function is still valid, QALoad now includes an improved option to automatically provide the same functionality. See [DO_DynamicCookieHandling](#) for details.

Syntax

```
DO_GetCookieFromReplyEx( const char *cookieName, char **cookieValue, char match );
```

Return Value

None.

Parameters

Parameter	Description
cookieName	String constant that specifies the name of the cookie to retrieve from the reply.
cookieValue	Address to a string where the cookie value is stored.
match	A wildcard character to use for regular expression matching before or after the cookie name. The default used by QALoad is the asterisk character (*).

Example

```
...
...
/* Declare Variables */
char *Cookie[1];
...
...
for(i=0;i<1;i++)
Cookie[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_DynamicCookieHandling(FALSE);
...
...
DO_GetCookieFromReplyEx("NUM", &Cookie[0], '*');
DO_SetValue("cookie000", Cookie[0]);
DO_Http("GET http://company.com/cgi-bin/cookiespipes.pl "
        "HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Host: company.com\r\n"
        "Cookie: HTMLA=FONTSIZE=LARGE; {*cookie000}; "
        "SITESERVER=ID=4b4ab9751bce9a95f74ec62\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(Cookie[i]);
Cookie[i]=NULL;
END_TRANSACTION();
```

...
...

DO_GetFormActionStatement

Applies to HTTP and SSL requests. Gets the ACTION tag from a requested form.

This feature is useful when a form dynamically changes what is stored in the ACTION tag.

Syntax

```
int DO_GetFormActionStatement( int nFormnum, char **ActionURL );
```

Return Value

1 for successful
0 for unsuccessful

Parameters

Parameter	Description
nFormnum	Specifies which form on a response to retrieve the ACTION tag from.
ActionURL	Address of the string where the ACTION tag will be stored.

Example

```
...
...
char *ActionURL[1];
...
...
for(i=0;i<1;i++)
ActionURL[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetFormActionStatement(Form (1), &ActionURL[0]);
DO_SetValue("action_statement0", ActionURL[0]);
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
"Content-Type: multipart/form-data; boundary="
"-----7d04c2740364\r\n"
"Host: company\r\n"
"Content-Length: {*content-length}\r\n"
"Cookie: username=anu; c2_LastVisit="
"Mon%20Mar%2013%0; c2_NumVisits=\r\n"
"Content-Disposition: form-data; name=\"entry \"\r\n\r\n\r\n"
"-----7d04c2740364\r\n\r\n");
...
...
for(i=0; i<1; i++)
{
free(ActionURL[i]);
}
END_TRANSACTION();
...
...
```

DO_GetFormValueByName

Applies to HTTP and SSL requests. Retrieves the value embedded in a form for the specified field.

This value can then be used subsequently in a call to the DO_SetValue command to pass it along to the CGI script associated with this form. DO_GetFormValueByName is generally seen when hidden fields are encountered in a form. QALoad's Convert facility automatically generates these commands for hidden fields.

Syntax

```
GetFormValueByName( int form_number, const char *field_type, const char *field_name, int
count, char **value );
```

Parameters

Parameter	Description
form_number	Integer specifying which form to search in an HTML document.
field_type	Type of field to search.
field_name	Name of the field to search.
count	If more than one field has the same name, a number specifying each field.
value	Address to a string where the result value will be stored.

Example

```
...
...
char *Field[2];
...
...
for(i=0;i<2;i++)
Field[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetFormValueByName(FORM (1), "hidden", "hidden", 1, &Field[0]);
DO_GetFormValueByName(FORM (1), "hidden", "hidden1", 1, &Field[1]);
...
...
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
...
...
BeginCheckpoint(); /* *FORM* */
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Host: company\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name }&{hidden}&{hidden1}&{submit}");
...
...
DO_HttpCleanup();
for(i=0; i<2; i++)
{
free(Field[i]);
}
...
```

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

DO_GetHeaderFromReply

Applies to HTTP and SSL requests. Retrieves the value of a header in the reply resulting from a DO_HTTP command.

Syntax

```
int DO_GetHeaderFromReply ( char *header, const char
*output_buffer, int nLength )
```

Return Value

1 for success
0 for unsuccessful

Parameters

Parameter	Description
header	A header to look for in the reply.
output_buffer	A string to store the result. Memory should already be allocated for it.
nLength	The length of space available in the output buffer.

Example

```
char OutputBuf[256];
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, */*\r\n"
        "Host: company.com\r\n"
        "Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID="
        "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
DO_GetHeaderFromReply("Content-Length:", OutputBuf, 255);
...
...
```

DO_GetLastHttpError

Applies to HTTP and SSL requests. Retrieves the integer indicating the error code of the last HTTP request sent with DO_Http.

Errors greater than 399 include the "Page not found" 404 error.

Syntax

```
int DO_GetLastHttpError();
```

Return Value

Returns the error code, or 0 if unsuccessful.

Parameters

None.

Example

```
int error;
char errorString[50];
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Host: company.com\r\n\r\n");
if ((error = DO_GetLastHttpError()) > 399)
{
    sprintf (errorString, "Error in response: %d\n", error);
    WWW_FATAL_ERROR ("DO_Http", errorString);
}
...
...
```

DO_GetRedirectedURL

Applies to HTTP requests. Modifies the parameter passed in for use in the next request.

This function is still supported, however, [DO_DynamicRedirectHandling](#) is preferred.

Syntax

```
int DO_GetRedirectedURL (char **URL)
```

Return Value

1 for successful

0 for unsuccessful

Parameters

Parameter	Description
URL	An address to a string.

Example

```
DO_Http(http_statement);

/* RedirectedURL[0]="http://company/cgi-bin/pm3D.htm"*/
DO_GetRedirectedURL(&RedirectURL[0]);

/* Request: 10 * From: QALoad WWW Capture Examples */
DO_SetValue("redirect_statement0", RedirectURL[0]);
DO_Http("GET {*redirect_statement0} HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://company/index.htm\r\n"
        "Accept-Language: en-us\r\n"
        "Accept-Encoding: gzip, deflate\r\n"
        "Host: company\r\n\r\n");
```

DO_GetReplyBuffer

Applies to HTTP and SSL requests. DO_GetReplyBuffer returns the HTTP response from the last DO_Http request.

Syntax

```
Const char * DO_GetReplyBuffer()
```

Return Value

The last HTTP reply or NULL if unsuccessful.

Parameters

None.

Example

```
const char * data;
...
...
BEGIN_TRANSACTION();
...
...

/* Request: 1 */

DO_Http("GET http://company.com/ HTTP/1.0\r\n\r\n" );
data = strstr(DO_GetReplyBuffer(), "data_key" );
if ( data == NULL )
{
  WWW_FATAL_ERROR ("DO_Http", "Data_key was missing in reply" );
}
...
...
```

DO_GetUniqueString


Applies to HTTP and SSL requests. Used to parse the most recent HTTP server reply to get the contents of a string that occurs between the left and right input strings.

Syntax

```
char *DO_GetUniqueString( const char *left, const char *right );
```

Return Value

The string (null-terminated) of characters between the left and right search strings provided as input. NULL If either the left or right search strings are not found.

 **Note:** DO_GetUniqueString allocates enough space to hold the string (including the NULL). Any memory created with the use of malloc results in a memory leak. Please remember to free any memory after using the returned string.

Parameters

Parameter	Description
left	A string containing the left search string.
right	A string containing the right search string.

Example

```
char *p;
char temp[1000];
...
...
DO_Http( "GET HTTP://www.yahoo.com HTTP/1.0\r\n"
        "Referer: HTTP://company/index.htm\r\n"
        "Proxy-Connection: Keep-Alive\r\n"
        "User-Agent: Mozilla/3.01 WinNT;I)\r\n"
        "Host: www.yahoo.com\r\n"
        "Accept: */*\r\n");
p = DO_GetUniqueString( "text to the left side of the string",
                      "text to the right side of the string" );

if ( p != NULL )
{
    strcpy( temp, p );
    free( p );
}
RR_printf( "String value = %s", temp );
```

DO_GetUniqueStringEx


Applies to HTTP and SSL requests. Used to parse a null-terminated input string (search) to get the contents of a string that occurs between the left and right input strings.

Syntax

```
char *DO_GetUniqueStringEx( const char *search, const char *left, const char *right );
```

Return Value

The string (null-terminated) of characters between the left and right search strings provided as input. NULL if either the left or right search strings are not found.

 **Note:** DO_GetUniqueStringEx allocates enough space to hold the string (including the NULL). Any memory created with the use of malloc results in a memory leak. Please remember to free any memory after the usage of the returned string.

Parameters

Parameter	Description
search	A string to be searched.
left	A string containing the left search string.
right	A string containing the right search string.

Example

```
char *p;
char temp[1000];
...
...
strcpy( temp, "Here is the search string." );
p = DO_GetUniqueStringEx( temp, "the", "string" );

if ( p != NULL )
{
    RR_printf( "String value = %s", p );
    free( p );
}
else
```

```
{
  RR_printf( "String not found" );
}
```

DO_Http

Applies to HTTP requests. Executes an HTTP request in the script.

DO_Http sends the request to the server. Any responses to the request are then processed by DO_Http and returned to the script. DO_Http returns text replies to the script.

Syntax

```
char *DO_Http( const char *http_statement );
```

Return Value

Character string containing the response from the server.

Parameters

Parameter	Description
http_statement	String containing a valid HTTP request to be sent to a server.

Example

```
...
...
DO_Http( "GET HTTP://www.yahoo.com HTTP/1.0\r\n"
  "Referer: HTTP://company/index.htm\r\n"
  "Proxy-Connection: Keep-Alive\r\n"
  "User-Agent: Mozilla/3.01 WinNT;I)\r\n"
  "Host: www.yahoo.com\r\n"
  "Accept: */*\r\n" );
...
...
```

DO_HttpCleanup

Applies to HTTP and SSL requests. Performs all necessary cleanup operations when a script exits or the user terminates the script.

 **Note:** This command is the same as DO_Clear (TRANSACTION).

Syntax

```
DO_HttpCleanup( );
```

Parameters

None.

Example

```
...
...
DO_Http( "GET HTTP://www.yahoo.com HTTP/1.0\r\n"
  "Referer: HTTP://company/index.htm\r\n"
  "Proxy-Connection: Keep-Alive\r\n"
  "User-Agent: Mozilla/3.01 WinNT;I)\r\n"
  "Host: www.yahoo.com\r\n"
  "Accept: */*\r\n" );
...
...
```

```

...
DO_HttpCleanup();
...
...
END_TRANSACTION();
...
...

```

DO_HttpVersion

Applies to HTTP and SSL requests. Specifies the version to use in the requests sent during playback.

This affects whether or not the replies may come back chunked. Only HTTP 1.1 requests receive chunked replies.

DO_HttpVersion is related to the HTTP Version Detection option on the WWW Advanced dialog box. From the Convert Options wizard, access the WWW Advanced dialog box by clicking the Advanced button. The default setting is Auto.

Syntax

```
DO_HttpVersion(const char *version);
```

Parameters

Parameter	Description
version	The HTTP version ("1.0", "1.1", or "Auto"). If specified as Auto, the version used for each request is determined from the request.


Example

```
DO_HttpVersion("Auto");
```

DO_InitHttp

Applies to HTTP and SSL requests. Also applies to Visual Scripting. Sets all necessary internal variables needed to load test an HTTP script.

Use this command at the beginning of every HTTP script, but never more than once in a script.

 **Note:** This function should be written exactly as shown below.

Syntax

```
DO_InitHttp(PLAYER_INFO *sInfo);
```

Parameters

Parameter	Description
sInfo	A pointer to a PLAYER_INFO memory structure.

Example

```

...
...
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
...
...
DO_InitHttp(s_info);

```

```

...
...
BEGIN_TRANSACTION( );
...
...
}

```

DO_IPSpooFEnable

Applies to HTTP and SSL requests. Enables each virtual user to appear to the web server as being sourced from a different network interface card.

This command is placed after the DO_InitHttp command. It is useful for those applications where the server keys off the originating IP address. To utilize this feature, the Player system must be configured with multiple static IP addresses. In addition, a local datapool file containing a list of valid IP addresses must be available to the Player. The Player tab on the QALoad Conductor Options dialog box provides an option for creating this local datapool file for NT-based Players.

The datapool file name defaults to using the datapool file pointed to by the qaload_ipspooF environment variable. This variable is automatically set when QALoad is installed on an NT-based system. Users of the UNIX-based Players must add this variable manually. The parameter to this command can be used to override the contents of the environment variable.

Syntax

```
Const char *DO_IPSpooFEnable(const char *filename);
```

Return Value

A string containing the IP address.

Parameters

Parameter	Description
Filename	String containing a fully qualified path name. This file contains a list of IP addresses to use. Set to "" to use the filename specified in the qaload_ipspooF environment variable.

Example

```

...
...
DO_IPSpooFEnable( "c:\\qaload\\ myipspooF.dat" );
BEGIN_TRANSACTION( );
...
...


```

DO_NTLMAuthorization

Applies to HTTP requests. Provides user ID and password (plain text or encrypted) information for NTLM authentication.

DO_NTLMAuthorization is related to the NTLM option on the QALoad Script Development Workbench Record Options wizard. When you select that option and enter user ID and password information, DO_NTLMAuthorization(string, string) is written to your script. QALoad attempts to use the user ID and password you entered to access the site. If the information is not accepted, QALoad reports the error and aborts.

At test time, when QALoad encounters a NTLM controlled site, it uses the NTLM user ID and password that are provided to access that site.

 **Note:** NTLM user names and passwords can be variablized by machine, but not by user.

Syntax

```
DO_NTLMAuthorization(const char *name, const char *password);
```


Parameters

Parameter	Description
name	A valid user ID for the NTLM-enabled site.
password	A valid password corresponding to the user ID.

Examples

Example 1:

```
...
...
BEGIN_TRANSACTION();
...
...
DO_NTLMAuthorization("user-id", "~encr~2038520348AKJAS");
...
...
END_TRANSACTION();
...
...
```

 **Note:** String must be enclosed in quotation marks (""), unless NULL is used.

Example 2:

When the user ID, password, and domain are provided:

```
DO_NTLMAuthorization("domain\\user_id", "~encr~506C205A545D");
```

Example 3:


When the domain is not provided:

```
DO_NTLMAuthorization("user_id", "~encr~506C205A545D");
```

Example 4:

When NULL is used and access is provided:

```
DO_NTLMAuthorization(NULL, NULL);
```

 **Note:** NULL is not enclosed in quotes.

DO_AdditionalSubRequest

Applies to HTTP and SSL requests. DO_AdditionalSubRequest manually adds a sub-request for the next DO_Http or DO_Https request. The request is specified as a URL.

Syntax

```
int DO_AdditionalSubRequest ( const char * szSubRequest );
```

Return Value

Returns the number of items in the pre-loaded sub-request list.

Parameters

Parameter	Description
szSubRequest	URL to add as a sub-request of the next DO_Http or DO_Https request.

Example

```
...
...
DO_AdditionalSubRequest ( "http://company.com/images/bar.gif" );
...
...
```

DO_ProxyAuthorization

Provides the username and password to access a password protected proxy server.

The password may be encrypted using QALoad's "~encr~" encryption. The username and password are inserted automatically as necessary during conversion. Note that you can variablize the username and password to emulate different users accessing the resources.

Syntax

```
DO_ProxyAuthorization(const char *username, const char *password);
```

Parameters

Parameter	Description
username	A valid user name for the resource you're attempting to access.
password	The associated password.

Example

```
DO_HttpVersion("Auto");
DO_SLEEP(2);

/* Request: 1 */

DO_ProxyAuthorization("smith", "~encr~0E636502080E");

BeginCheckpoint(" http://iris/redline - chkpt: 1");

DO_Http("GET http://iris/redline HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, "
"application/vnd.ms-excel, application/msword, "
"application/vnd.ms-powerpoint, */*\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT; CPWR)\r\n"
"Host: iris\r\n\r\n"
);
```

DO_ProxyExceptions

Applies to HTTP and SSL requests. Tells QALoad not to use the proxy server for hosts in the proxy exceptions list, so you can replay requests both inside and outside of the firewall in the same script.

This command is written to the script when the option Automatically configure proxy options and launch browser is selected on the QALoad Script Development Workbench Record Options wizard.

DO_ProxyExceptions is written to the script between BEGIN_TRANSACTION and the first request.

Syntax

```
int DO_ProxyExceptions(const char *list);
```

Return Value

-1 if the list is NULL.
0 if successful.

Parameters

Parameter	Description
list	List of proxy addresses in exceptions list. Note that addresses are separated by commas in the script.

Example

```
...
...
BEGIN_TRANSACTION();
...
...
DO_UseProxy ("internet.company.com:80" );
DO_SSLUseProxy ("internet.company.com:90" );
DO_ProxyExceptions("company.sample.com, "company2.company.com" );
...
...
```

DO_SaveReplyType

Applies to HTTP and SSL requests. Specifies types of replies to save.

Normally, only replies returned from the server whose type begin with "text/" are saved. Use DO_SaveReplyType to specify which type(s) to save. You can specify multiple types if you separate them with a semi-colon (;).

In a reply, the type is specified in the "Content-Type:" tag. You access the reply by saving a pointer returned from the DO_Http command:

```
char *p;
...
p = DO_Http("GET http://www.nosuch.com/...");
```

Syntax

```
DO_SaveReplyType(const char *types);
```

Parameters

Parameter	Description
types	Reply types to save (for example, "text;image/gif" saves replies specified as text or image/gif in the replies' "Content-Type" tag).


Example

```
...
...
DO_SaveReplyType("text;image/gif");
BEGIN_TRANSACTION();
...
...
```

DO_SetAssumedContentType

Applies to HTTP and SSL requests. Sets the default content type if the web server doesn't send a content-type header.

If any reply from a web server doesn't contain a content-type header, then QALoad assumes the content-type is application/octet-stream. application/octet-stream is not processed by QALoad and the body of such a reply is not available. To override the default assumed content-type, use this function to set a new content type.

 **Note:** According to the HTTP specification, returning a response without a content-type is undefined behavior and may indicate a problem on the server.

Setting the assumed content type to text/html allows the reply to be treated as an HTML document.

Once you have set the assumed content type, it does not change until the next call to DO_SetAssumedContentType.

This command corresponds to the Assumed Content-Type field on the QALoad Script Development Workbench Record Options wizard.

Syntax

```
DO_SetAssumedContentType(const char *ContentType);
```

Parameters

Parameter	Description
ContentType	The mime type that is used as the new default content type if the web server doesn't send a content type header.

Example

```
DO_SetAssumedContentType("text/html");
```

DO_SetBaudRate

See also [WWW](#)

Applies to HTTP and SSL requests. Causes a virtual user to delay transmission and reception of network traffic to emulate a given modem speed. Returns the baud rate the virtual user will use.

Syntax

```
int DO_SetBaudRate(int nBaud)
```

Return Value

Returns the baud rate the virtual user will use.

Parameters

Parameter	Description
nBaud	The rate the virtual user will use. If nBaud is set to 0, modem emulation is shut off.

Example

```
...
...
BEGIN_TRANSACTION();
DO_SetBaudRate(28800);
```

...
...

DO_SetBaudRateEx

Applies to HTTP and SSL requests. Causes a virtual user to delay transmission and reception of network traffic to emulate a given modem speed. The transmission rate and the reception rate are set as separate values.

Syntax

```
int DO_SetBaudRateEx (int nTransmissionRate, int nReceptionRate)
```

Return Value

Returns the transmission rate the virtual user will use.

Parameters

Parameter	Description
nTransmissionRate	The transmission rate the virtual user will use. If nTransmissionRate is set to 0, modem transmission emulation is shut off.
nReceptionRate	The reception rate the virtual user will use. If nReceptionRate is set to 0, modem reception emulation is shut off.

Example

```
...
...
BEGIN_TRANSACTION();
DO_SetBaudRateEx(28800, 36600);
...
...
```

DO_SetCheckpointName

Sets the name of the next automatic checkpoint for the next DO_Http or DO_Https statement in the script.

Syntax

```
void DO_SetCheckpointName (const char *szCheckpointName);
```

Parameters

Parameter	Description
szCheckpointName	The name for the next automatic checkpoint.

Example

```
DO_SetCheckpointName("Login to Website");
DO_Https("POST https://dbhost.company.com/login.asp HTTP/1.1\r\n"
"\r\n"
"{domain}&{username}&{password}");
```

DO_SetCookie

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

The path of the cookie is "/". The domain of the cookie is the same as the next DO_Http or DO_Https request. If you wish to set a particular domain or path, use DO_SetCookieEx.

Once a cookie is set, it remains for the rest of the transaction. To remove the cookie, use DO_SetCookieEx with the name of the cookie to remove and an expiration value of -1.

DO_SetCookie requires DO_DynamicCookieHandling to be set to TRUE.

Syntax

```
BOOL DO_SetCookie ( const char * szName, const char * szValue );
```

Return Value

TRUE for successful
 FALSE for unsuccessful

Parameters

Parameter	Description
szName	Name of the cookie to set.
szValue	Value of the cookie to set.

Example

```
...
...
BEGIN_TRANSACTION();
...
...
DO_SetCookie ( "cookie1", "desired value" );
/* Request: 1 */
/*
 * This request will have "cookie1" sent with this request
 */
DO_Http ( "GET http://company.com/ HTTP/1.0\r\n\r\n" );
...
...
```

DO_SetCookieEx

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

Once a cookie is set, it remains for the rest of the transaction. To remove the cookie, use DO_SetCookieEx with the name of the cookie to remove and a max age of -1.

DO_SetCookie requires DO_DynamicCookieHandling to be set to TRUE.

Syntax

```
BOOL DO_SetCookieEx ( const char * szName, const char * szValue,
                    const char * szDomain, const char * szPath,
                    int nMaxAge, BOOL bSecure );
```

Return Value

TRUE for successful
 FALSE for unsuccessful

Parameters

Parameter	Description
szName	Name of the cookie to set.
szValue	Value of the cookie to set.
szDomain	Domain of the cookie. The domain of the cookie controls what hosts the cookie is sent to.
szPath	The path of the cookie. The path of the cookie controls when a cookie is sent to a host based on the path of the URL.
nMaxAge	Time to live of the cookie. Use a value of 0 for a session cookie and -1 for an expired cookie.
bSecure	Boolean flag (TRUE or FALSE). If the value is TRUE, then the cookie only is sent with SSL request. If the value is FALSE, then the cookie is sent with HTTP and SSL requests.

Example

```

...
...
BEGIN_TRANSACTION();
...
...
DO_SetCookieEx ( "cookie1", "desired value", ".company.com", "/", 1000, FALSE );
/* Request: 1 */
/*
 * This request will have "cookie1" sent with this request
 */
DO_Http ( "GET http://company.com/ HTTP/1.0\r\n\r\n" );
...
...

```

DO_SetJavascriptCleanupThreshold

Applies to HTTP and SSL requests. Periodically QALoad destroys its internal JavaScript model and recreates it.

DO_SetJavascriptCleanupThreshold sets a count of the number of times JavaScript parsing is done before destroying and recreating the model. By default, the count is 300.

Cleaning up JavaScript takes CPU time, and the Javascript model takes up more memory the longer the same model is used. To reduce CPU usage, set the count higher. To reduce the memory footprint, set the count lower.

Syntax

```
DO_SetJavascriptCleanupThreshold(int nThreshold)
```

Parameters

Parameter	Description
nThreshold	Number of JavaScript evaluations to make before cleaning up the JavaScript engine.

Example

```
...
...
DO_SetJavascriptCleanupThreshold(200);
...
...
```

DO_SetMaxBrowserThreads

Applies to HTTP and SSL requests. Specifies the number of concurrent connections to make for playback.

This command relates to the Max Concurrent Connections option on the WWW Advanced options dialog box. The value you enter in that field is inserted in the script.

Syntax

```
DO_SetMaxBrowserThreads(int count);
```

Parameters

Parameter	Description
count	The number of connections to make. QALoad accepts 1-8. The default is 2.

Example

```
BEGIN_TRANSACTION();
DO_SetMaxBrowserThreads(2);
```

DO_SetMaximumRetries

Applies to HTTP and SSL requests. Sets the maximum number of times a virtual user should attempt to retrieve a graphic or page that failed.

Similar to the behavior of Netscape and Internet Explorer.

Syntax

```
DO_SetMaximumRetries(int nValue)
```

Parameters

Parameter	Description
nValue	The default is 4.

Example

```
...
...
BEGIN_TRANSACTION();
DO_SetMaximumRetries(5);
...
...
```

DO_SetRefreshTimeout

Specifies how long to wait for a meta refresh or an HTTP refresh header.

The HTML meta tag can set a number of seconds before a refresh. When that number of seconds has expired, then the browser loads the URL specified in the meta refresh.

QALoad's WWW replay only refreshes the page if the number of seconds specified in the refresh is less than or equal to the timeout value set by `DO_SetRefreshTimeout`. If the refresh is set too large, then QALoad's WWW replay can get stuck in an infinite loop.

Syntax

```
int DO_SetRefreshTimeout(int nTimeout);
```

Parameters

Parameter	Description
nTimeout	How many seconds to wait for a refresh, the default is 0.

DO_SetRetryWait

Applies to HTTP and SSL requests. Sets the delay between retries in seconds.

Syntax

```
DO_SetRetryWait(int nValue)
```

Parameters

Parameter	Description
nValue	Delay between retries, in seconds. Default is 1.

Example

```
DO_SetRetryWait(6);
```

DO_SetTimeout

Applies to HTTP and SSL requests. Specifies how long to wait for a reply from the server. If a reply is not received within the specified time, the virtual user will fail with a fatal error.

`DO_SetTimeout` allows you to more closely emulate browser behavior when requests go unanswered due to server or network problems. Normally a browser would wait until it receives a reply or the user cancels the request by clicking the Stop button.

This command relates to the Server Response Timeout option on the WWW Advanced options dialog box. The default is 120 seconds.

Syntax

```
DO_SetTimeout(int timeout);
```

Parameters

Parameter	Description
timeout	The number of seconds to wait. The default is 120.

Example

```
DO_SetTimeout(120); /* Maximum time to wait for an HTTP Reply */
```

DO_UseEntityList

Applies to HTTP and SSL requests. Decodes non-ASCII character entities.

Syntax

```
void DO_UseEntityList ( ENTITY_LIST );
```

Parameters

Parameter	Description
ENTITY_LIST	User-defined Entity list.

Example

For examples and more information about this command, see [HTML character entities and numeric references](#).

DO_UseNumericReferenceList

Applies to HTTP and SSL requests. Decodes non-ASCII numeric references.

Syntax

```
void DO_UseNumericReferenceList ( NUMERIC_REFERENCE_LIST );
```

Parameters

Parameter	Description
NUMERIC_REFERENCE_LIST	User-defined Numeric Reference list.

Example

For examples and more information about this command, see [HTML character entities and numeric references](#).

DO_UsePersistentConnections

Applies to HTTP and SSL requests. Turns the use of persistent connections on or off.

It will always terminate the current persistent connection if one is present. This will allow persistent connections to be reset in transaction loops to better simulate a real user test.

Syntax

```
void DO_UsePersistentConnections ( BOOL bEnable )
```

Return Value

None.

Parameters

Parameter	Description
bEnable	A flag indicating if the Use Persistent Connections option should be enabled (1=TRUE, 0=FALSE).

Example

```

...
...
BEGIN_TRANSACTION();
DO_UsePersistentConnections(1);
...
...

```

DO_UseProxy

Applies to HTTP and SSL requests. Specifies a proxy server to use during testing.

If you select the Use a proxy server option on the QALoad Script Development Workbench 's Record Options wizard before you record, a DO_UseProxy command is inserted at the beginning of your script. If you change your proxy server while recording, QALoad 's Record facility detects the modification and inserts another DO_UseProxy() into the script.

Syntax

```
int DO_UseProxy( const char *proxy );
```

Return Value

Always returns 0.

Parameters

Parameter	Description
proxy	String containing the proxy server and port separated by a colon.

Example

```

...
...
BEGIN_TRANSACTION();
...
...
DO_UseProxy ( "internet:80" );
DO_SSLUseProxy ( "internet.company.com:90" );
DO_ProxyExceptions( "company.sample.com, "company2.company.com" );
...
...

```

DO_UseProxyAutomaticConfiguration

Applies to HTTP and SSL requests. Downloads the proxy automatic configuration (PAC) script at the specified URL. The rest of the transaction will use the PAC script to determine which proxy, if any, to connect to hosts.

Syntax

```
BOOL DO_UseProxyAutomaticConfiguration ( const char * szUrl );
```

Return Value

TRUE if successful, else FALSE.

Parameters

Parameter	Description
-----------	-------------

szUrl	URL where the proxy automatic configuration script is located.
-------	--

Example

```

...
...
BEGIN_TRANSACTION();
DO_UseProxyAutomaticConfiguration( "http://proxy config.host.com/" );
...
...
/*Request: 1*/
/*
*The PAC script downloaded from http://proxyconfig.host.com/
*determine what proxy, if any, to use to connect to
*company.com
*/
DO_Http ( "GET http://company.com/ HTTP/1.0\r\n\r\n" );
...
...

```

DO_VerifyDocTitle

Applies to HTTP and SSL requests. Compares the parameters and match type passed in the parameters against the HTML page title specified in the response received from the HTTP request.

Syntax

```
int DO_VerifyDocTitle ( const char *szTitle, int nType ) ;
```

Return Value

The function will return an integer value of either 0 or 1. If a match is found (1), the Player debug window will indicate so. If not (0), the function will call WWW_FATAL_ERROR which will either abort the test or continue, based upon the ABORT_ON_ERROR flag.

Parameters

Parameter	Description
szTitle	A character string specifying a title to search for in the HTTP response. This is generated by Convert using the entire document title, the title prefix, or the title suffix, as specified on the QALoad Script Development Workbench Convert Options wizard.
nType	This parameter should be one of three values: TITLE, PREFIX, or SUFFIX, corresponding to the comparison options available on the QALoad Script Development Workbench Convert Options wizard.


Example

```
DO_Http ( http_statement ) ;
DO_VerifyDocTitle ( "Welcome to Compuware" , TITLE ) ;
```

DownloadMediaFromASX

Applies to Windows Media Player streaming media. Dynamically parses an ASX file from the previous response and initiates and waits for completion of the specified Windows Media resources download.

DownloadMediaFromASX is a deprecated function. Use a combination of the Click_On function with the PlayMedia function instead.

 **Note:** For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad" chapter of the *QACenter Performance Edition Installation and Configuration Guide*.

Syntax

```
DownloadMediaFromASX( int secDuration );
```

Parameters

Parameter	Description
secDuration	Specifies the number of seconds of media to download. Specifying 0 means read the entire media.

Example

```
Do_Http("GET http://host/test.asx HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, image/jpeg, image/"
        "pjpeg,application/vnd.ms-excel, application/"
        "vnd.ms-powerpoint, msword, */*\r\n"
        "Accept-Language: en-us\r\n"
        "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows"
        "NT 5.0)\r\n\r\n" );

// Play the media file(s) specified in the ASX file for 50 seconds.
DownloadMediaFromASX(50);
```

DownloadMediaRP

Applies to Real Networks Streaming Media. Initiates and waits for completion of the specified multi-media resource download.

DownloadMediaRP is a deprecated function. Use a combination of the Click_On function with the PlayMedia function instead.

 **Notes:**

- ! Enable streaming media download by selecting the Streaming Media check box on the WWW Advanced Universal Convert Options dialog box in the QALoad Script Development Workbench.
- ! Real Networks streaming media is only supported in process mode. On the QALoad Player main window, in the **Run As:** group, select the **Process** option.
- ! For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad" chapter of the *QACenter Performance Edition Installation and Configuration Guide*.


Syntax

```
DownloadMediaRP( char *URL, int timeout );
```

Parameters

Parameter	Description
URL	Specifies the location (in URL format) of the streaming media file.
timeout	Specifies the number of seconds of media to play. Specify 0 (the default in

the script) to play the entire media transaction. Specify another number, such as a value of 10, to have the timeout buffer the media and play the clip for 10 seconds.

 **Note:** This timeout refers to clip time. The elapsed time of a Real Networks media transaction will likely be longer than the timeout.


Example

```
DownloadMediaRP("http://host:8099/ramgen/realvideo.rm", 0);
DownloadMediaRP("http://host2:8080/realmp3.mp3", 10);
```

DownloadMediaWMP

Applies to Windows Media Player streaming media. Initiates and waits for completion of the specified Windows Media resource download.

DownloadMediaWMP is a deprecated function. Use a combination of the [Click_On](#) function with the [PlayMedia](#) function instead.

 **Note:** For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad " chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

```
DownloadMediaWMP( char *reqURL, int secDuration );
```

Parameters

Parameter	Description
reqURL	Specifies the location (in URL format) of the streaming media file.
secDuration	Specifies the number of seconds of media to download. Specify 0 to read the entire media file.

Example

```
// Requests welcome2.asf from qacmedia over TCP ("mmst://")
// Play the file for 10 seconds
DownloadMediaWMP("mmst://qacmedia/welcome2.asf", 10 );
```

EnableStatisticsRP

Applies to Real Networks Streaming Media. Enables capture of media player performance statistics during a load test. Compuware recommends that this function is called in the initial section of a Web script, before the SYNCHRONIZE() call. Although it can be called at any point in the script, this command must appear in the script prior to any DownloadRPMedia call.

Notes:

- ! Exercise caution when using this feature. Real Networks streaming media uses extra system resources and may degrade performance or skew test results.
- ! By default, capturing statistics is not enabled.

- ! Real Networks streaming media is only supported in process mode. On the QALoad Player main window, in the Run As: group, select the Process option.
- ! For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad " chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

```
EnableStatisticsRP( int flags, int interval, BOOL traceOutput );
```

Parameters

Parameter	Description
flags	<p>Determines which statistics to show. The flag values in the following table can be combined using a logical OR. Flag values include:</p> <p>QAL_WWW_RN_STAT_ALL_LEVELS: All statistic levels QAL_WWW_RN_STAT_PLAYER: Media Player level statistics QAL_WWW_RN_STAT_SOURCE: Source level statistics QAL_WWW_RN_STAT_STREAM: <not implemented> QAL_WWW_RN_STAT_ALL: Enable all levels, all counters QAL_WWW_RN_STAT_PLAYER_ALL: All Media Player level statistics QAL_WWW_RN_STAT_SOURCE_ALL: All source level statistics QAL_WWW_RN_STAT_STREAM_ALL: All stream level statistics QAL_WWW_RN_STAT_ALL_COUNTERS: All counters QAL_WWW_RN_STAT_NORMAL_PKTS: Packets not lost, late, etc. QAL_WWW_RN_STAT_RECOVERED_PKTS: Packets recovered QAL_WWW_RN_STAT_RECEIVED_PKTS: Packets received QAL_WWW_RN_STAT_LOST_PKTS: Packets currently lost QAL_WWW_RN_STAT_LATE_PKTS: Late packets QAL_WWW_RN_STAT_CLIP_BANDWIDTH: Bandwidth at which the clip was encoded QAL_WWW_RN_STAT_AVE_BANDWIDTH: Average bandwidth so far QAL_WWW_RN_STAT_CUR_BANDWIDTH: Current bandwidth</p>
interval	Report every nth stat received.
traceOutput	TRUE means send enabled stats to QALoad Player window (if QALoad Player window output is enabled).

Example

```
// Records, current bandwidth, average bandwidth, and the clip
// bandwidth at the Player (media player) level as often as
// the statistics are updated.

EnableStatisticsRP( QAL_WWW_RN_STAT_PLAYER
                   QAL_WWW_RN_STAT_AVE_BANDWIDTH
                   QAL_WWW_RN_STAT_CLIP_BANDWIDTH
                   QAL_WWW_RN_STAT_CUR_BANDWIDTH,
                   0, TRUE );
```

Fill_In

Applies to Visual Scripting. Used to represent how the user filled in fields on a form before clicking on a submit button. The values that are passed to Fill_In are expected to be plain text with no encoding other than using + to join multiple selects for LIST_BOX.

Prototypes

```
boolean Fill_In ( enumeration control_type, string description, string value );
boolean Fill_In ( enumeration control_type, enumeration specifier,
                 string description, string value );
boolean Fill_In ( enumeration control_type, integer count, enumeration specifier,
                 string description, string value );
boolean Fill_In ( enumeration control_type, integer count, string value );
```

Return Value

Returns true if the value was stored properly. Otherwise, returns false.

Parameters

Parameter	Description																
control_type	<p>The type of link to try to click on. The control types include:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ATTACH_FILE</td> <td>Attach a file with the standard attach file control.</td> </tr> <tr> <td>CHECK_BOX</td> <td>Select or clear a check box.</td> </tr> <tr> <td>HIDDEN</td> <td>Set the value of a hidden control (no browser equivalent).</td> </tr> <tr> <td>LIST_BOX</td> <td>Select an element from a drop-down or multi-select list.</td> </tr> <tr> <td>RADIO_BUTTON</td> <td>Push a radio button.</td> </tr> <tr> <td>TEXT_BOX</td> <td>Type text into a text box.</td> </tr> </tbody> </table>	Type	Description	ATTACH_FILE	Attach a file with the standard attach file control.	CHECK_BOX	Select or clear a check box.	HIDDEN	Set the value of a hidden control (no browser equivalent).	LIST_BOX	Select an element from a drop-down or multi-select list.	RADIO_BUTTON	Push a radio button.	TEXT_BOX	Type text into a text box.		
Type	Description																
ATTACH_FILE	Attach a file with the standard attach file control.																
CHECK_BOX	Select or clear a check box.																
HIDDEN	Set the value of a hidden control (no browser equivalent).																
LIST_BOX	Select an element from a drop-down or multi-select list.																
RADIO_BUTTON	Push a radio button.																
TEXT_BOX	Type text into a text box.																
description	The text of the link, or the appropriate text expected by a specifier.																
specifier	<p>The way the text is used to find the link. The specifier types include:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>AFTER</td> <td>Fill_In link after unique HTML code in the page.</td> </tr> <tr> <td>ALT_ATTRIBUTE</td> <td>The alt attribute of the HTML tag.</td> </tr> <tr> <td>BEFORE</td> <td>Fill_In link before unique HTML code in the page.</td> </tr> <tr> <td>CONTAINING</td> <td>Fill_In link that contains unique HTML code in the page.</td> </tr> <tr> <td>DESCRIPTION</td> <td>The description as seen in the browser (default).</td> </tr> <tr> <td>NAME_ATTRIBUTE</td> <td>The name attribute of the HTML tag.</td> </tr> <tr> <td>SRC_ATTRIBUTE</td> <td>The src attribute of the HTML tag.</td> </tr> </tbody> </table>	Type	Description	AFTER	Fill_In link after unique HTML code in the page.	ALT_ATTRIBUTE	The alt attribute of the HTML tag.	BEFORE	Fill_In link before unique HTML code in the page.	CONTAINING	Fill_In link that contains unique HTML code in the page.	DESCRIPTION	The description as seen in the browser (default).	NAME_ATTRIBUTE	The name attribute of the HTML tag.	SRC_ATTRIBUTE	The src attribute of the HTML tag.
Type	Description																
AFTER	Fill_In link after unique HTML code in the page.																
ALT_ATTRIBUTE	The alt attribute of the HTML tag.																
BEFORE	Fill_In link before unique HTML code in the page.																
CONTAINING	Fill_In link that contains unique HTML code in the page.																
DESCRIPTION	The description as seen in the browser (default).																
NAME_ATTRIBUTE	The name attribute of the HTML tag.																
SRC_ATTRIBUTE	The src attribute of the HTML tag.																
count	The nth match. For example, if a Web page has three text boxes with the same description, such as Submit, use a count of 3 to match the third button.																
value	The value to insert into the control.																

Examples

```
Fill_In ( TEXT_BOX, "Name:" "Jeff" );
Fill_In ( CHECK_BOX, NAME_ATTRIBUTE, "timed", "check" );
Fill_In ( ATTACH_FILE, 1, "c:\\MyFiles\\Datafile.dat" );
```

Get

Applies to Visual Scripting. Retrieves data from the virtual browser. Whole pages, specific frames, and text strings from within the document can be retrieved.

Prototypes


```
page_id Get ( enumeration type );
page_id Get ( enumeration type, string description );
page_id Get ( enumeration type, string description, integer count );
page_id Get ( enumeration type, enumeration specifier, string description );
page_id Get ( enumeration type, enumeration specifier, string description, integer count );
page_id Get ( enumeration type, integer count );
integer Get ( enumeration type, enumeration specifier );
string Get ( enumeration type, enumeration specifier );
string Get ( enumeration type, enumeration specifier, string left, string right );
string Get ( enumeration type, enumeration specifier, string xpath-string );
```

Return Value

Returns a value specified by the return type of the type parameter upon success. If Get fails, it returns NULL or zero.

Parameters

Parameter	Description																				
type	<p>The item to get. Item types are listed in the following table:</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> <th>Return Type</th> <th>Parameters</th> </tr> </thead> <tbody> <tr> <td>REPLY</td> <td>Returns the entire reply buffer</td> <td>string or integer</td> <td>specifier</td> </tr> <tr> <td>PAGE</td> <td>Return the current page</td> <td>page_id</td> <td>None</td> </tr> <tr> <td>FRAME</td> <td>Return the specified frame</td> <td>page_id</td> <td>Same as Click_On: link, description, specifier, count</td> </tr> <tr> <td>XML</td> <td>Return value for the XML node specified in XPath-type syntax.</td> <td>string</td> <td>specifier, xpath-string</td> </tr> </tbody> </table>	Type	Description	Return Type	Parameters	REPLY	Returns the entire reply buffer	string or integer	specifier	PAGE	Return the current page	page_id	None	FRAME	Return the specified frame	page_id	Same as Click_On: link, description, specifier, count	XML	Return value for the XML node specified in XPath-type syntax.	string	specifier, xpath-string
Type	Description	Return Type	Parameters																		
REPLY	Returns the entire reply buffer	string or integer	specifier																		
PAGE	Return the current page	page_id	None																		
FRAME	Return the specified frame	page_id	Same as Click_On: link, description, specifier, count																		
XML	Return value for the XML node specified in XPath-type syntax.	string	specifier, xpath-string																		
description	The text to find as expected by the specifier.																				
specifier	<p>The way to do the specified get. The specifiers are listed in the following tables:</p> <p>Frame Specifiers</p> <table border="1"> <thead> <tr> <th>Specifier</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>NAME_ATTRIBUTE</td> <td>Name attribute of the frame (default.)</td> </tr> </tbody> </table>	Specifier	Description	NAME_ATTRIBUTE	Name attribute of the frame (default.)																
Specifier	Description																				
NAME_ATTRIBUTE	Name attribute of the frame (default.)																				

	SRC_ATTRIBUTE	Src attribute of the frame.		
	CONTAINING	Return the frame that contains the given HTML source in the page.		
	BEFORE	Return the frame before the given HTML source in the page.		
	AFTER	Return the frame after the given HTML source in the page.		
Reply Specifiers				
	Specifier	Description	Return Type	Parameters
	ENTIRE_BUFFER	Return the entire reply buffer (default)	string	None
	HTTP_STATUS	Return the HTTP status code	integer	None
	HTTP_HEADER	Return a HTTP header	string	string
	STRING	Return the text between the left and right parameters	string	string, string
	TAG	Return an entire tag ( Note: Causes extra processing)	string	string or string, integer
XML Specifiers				
	Specifier	Description	Return Type	Parameters
	TEXT	The value between tags of last element in XPath (default)	string	string
	ATTRIBUTE	The value of specified attribute of last element in XPath.	string	string
count	The nth item to match.			
left	The text to the left side of desired text.			
right	The text to the right side of desired text.			
xpath-string	The XPath string to match.			

Example

```

page = Get (PAGE);
page = Get (FRAME, "table of contents");
page = Get (FRAME, "sidebar", 2);
page = Get (FRAME, CONTAINING, "<frame longdesc='the way down the road'");
page = Get (FRAME, SRC_ATTRIBUTE, "fun.html", 2);
page = Get (FRAME, 2);
str = Get (REPLY, ENTIRE_BUFFER);
i = Get (REPLY, HTTP_STATUS);
str = Get (REPLY, STRING, "<body special=' ', ' '");
str = Get (REPLY, TAG, "soap-envelope");
str = Get (REPLY, TAG, "img", 5);
    
```



```
str = Get (XML, TEXT,
"/SOAP-ENV:Envelope/SOAP-ENV:Body/SOAPSDK1:EchoIntResponse/Result");
```

Navigate_To

Applies to Visual Scripting. Reads a URL typed in the Web browser's address field and constructs a request to navigate to the URL, or reads another request typed in the browser's address field, finishes the request and navigates to the request. `Navigate_To` is a direct replacement for `DO_Http`.

Prototypes

```
boolean Navigate_To ( string URL );
boolean Navigate_To ( string URL, enumeration encoding);
```

Return Value

Returns true if the requested page is successfully retrieved. Otherwise, returns false.

Parameters

Parameter	Description						
URL	A URL containing the location of the page to be requested.						
encoding	The encoding format for any CGI parameters of the URL. Valid values are: <table border="1" data-bbox="527 913 1502 1144"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>USE_FORM_ENCODING</td> <td>Encode parameters like form controls.</td> </tr> <tr> <td>USE_LINK_ENCODING</td> <td>Encode parameters like anchor CGI parameters.</td> </tr> </tbody> </table>	Value	Description	USE_FORM_ENCODING	Encode parameters like form controls.	USE_LINK_ENCODING	Encode parameters like anchor CGI parameters.
Value	Description						
USE_FORM_ENCODING	Encode parameters like form controls.						
USE_LINK_ENCODING	Encode parameters like anchor CGI parameters.						

Examples


```
Navigate_To ( "http://server.com/" );
Navigate_To ( "GET http://server.com/HTTP/1.1\r\nBizzare-Header: just for kicks\r\n\r\n" );

// Encode CGI parameters as if the url is an anchor
Set(NEXT_REQUEST_ONLY, CGI_PARAMETER, "name", "John Doe");
Navigate_To("http://host.com/anchor.pl", USE_LINK_ENCODING);
// request: http://host.com/anchor.pl?name=John%20Doe

// Encode CGI parameters as if the url is a form request
Set(NEXT_REQUEST_ONLY, CGI_PARAMETER, "name", "John Doe");
Navigate_To("http://host.com/form.pl", USE_FORM_ENCODING);
// request: http://host.com/form.pl?name=John+Doe
```

PlayMedia


Applies to Real Networks and Windows streaming media. Initiates and plays back the streaming media file that was stored in a previous call to the `Click_On` function.

 **Notes:** Real Networks streaming media is only supported in process mode. On the QALoad Player main window, in the **Run As:** group, select the **Process** option. For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad" chapter of the *QACenter Performance Edition Installation and Configuration Guide*.

Syntax

```
PlayMedia( int timeout );
```

Parameters

Parameter	Description
timeout	<p>The maximum amount of time to play back the requested streaming media file. A value of 0 indicates that the entire file should be played.</p> <p> Note: This timeout refers to clip time. The elapsed time of a Real Networks media transaction will likely be longer than the timeout.</p>

Example

```
//Play the file for 10 seconds
PlayMedia(10);
```

Post_To

Applies to Visual Scripting. Reads a URL typed in the Web browser's address field as well as the encoding type. It then constructs a request to send a post to the URL.


Prototypes

```
boolean Post_To ( string URL );
boolean Post_To ( string URL, string content-type );
boolean Post_To ( string URL, enumeration encoding );
```

Return Value

Returns true if the requested page is successfully retrieved. Otherwise, returns false.

Parameters

Parameter	Description										
URL	A URL containing the location of the page to be requested.										
content_type	The content type of the data being posted. (DEFAULT, MULTIPART_FORM_DATA, or TEXT_PLAIN).										
encoding	<p>The method attribute of the HTML form tag. Valid values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>MULTIPART_FORM_DATA</td> <td>MIME style posting used for attaching files.</td> </tr> <tr> <td>TEXT_PLAIN</td> <td>"name=plain" (rarely used format)</td> </tr> <tr> <td>WWW_FORM_URLENCODED</td> <td>"name=value& ", name and value encoded</td> </tr> <tr> <td>MISSING_CONTENT_TYPE</td> <td>Use to handle the rare condition of a POST without a content type. POST data must be pre-encoded.</td> </tr> </tbody> </table> <p> Note: The HTTP 1.1 specification requires a content type header with every POST.</p>	Value	Description	MULTIPART_FORM_DATA	MIME style posting used for attaching files.	TEXT_PLAIN	"name=plain" (rarely used format)	WWW_FORM_URLENCODED	"name=value& ", name and value encoded	MISSING_CONTENT_TYPE	Use to handle the rare condition of a POST without a content type. POST data must be pre-encoded.
Value	Description										
MULTIPART_FORM_DATA	MIME style posting used for attaching files.										
TEXT_PLAIN	"name=plain" (rarely used format)										
WWW_FORM_URLENCODED	"name=value& ", name and value encoded										
MISSING_CONTENT_TYPE	Use to handle the rare condition of a POST without a content type. POST data must be pre-encoded.										

Examples

```
Post_To ( "http://server.com/form.pl" );
Post_To ( "http://server.com/file-upload.pl", MULTIPART_FORM_DATA );

// When posting content that is not application/x-www-form-urlencoded,
// multipart/form-data, or text/plain.
Set(NEXT_REQUEST_ONLY, POST_DATA, "RAW DATA\r\nPOSTED AS IS");
Post_To("http://host.com/custom.pl", "application/octet-stream");
```

RandNumString

Applies to Visual Scripting. Generates a random number from minimum to maximum.

Prototypes

```
string RandNumString ( int minimum, int maximum );
```

Return Value

Returns the generated random number as a string.

Parameters

Parameter	Description
minimum	The lower bound of the random number.
maximum	The upper bound of the random number.

Examples

```
RandNumString ( 20, 500 );
```

Region

Applies to Visual Scripting. Marks the `region_number` parameter as an image map region.

Prototypes

```
string Region ( int region_number );
```

Return Value

Returns the region number as a string.

Parameters

Parameter	Description
<code>region_number</code>	The region number.

Examples

```
// Region returns the string passed into it. It is a label to make
// clicking on a client side image map easier to read.
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/client-map.jpg", Region("2"));

// does the same as
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/client-map.jpg", "2");
```

RESTART_TRANSACTION_BOTTOM

Applies to Visual Scripting. Used to define a point at the end of the transaction for anything that needs to be deallocated or uninitialized. When transaction restarting occurs for a failed transaction, QALoad will first execute any code starting after the call to `RESTART_TRANSACTION_BOTTOM` allowing you to clean up important information and prevent memory leaks before retrying the transaction.

Syntax

```
RESTART_TRANSACTION_BOTTOM() ;
```

Parameters

None.

Example

```
BEGIN_TRANSACTION() ;
RESTART_TRANSACTION_TOP() ;
TRANSACTION CODE...
RESTART_TRANSACTION_BOTTOM() ;
DO_HttpCleanup() ;
DO_SomeOtherMiddlewareCleanup() ;
END_TRANSACTION() ;
```

RESTART_TRANSACTION_TOP

Used to define a point at the beginning of the transaction loop that QALoad can use to rewind the transaction if the transaction fails and Restart Transaction error handling has been selected in the QALoad Conductor.

To enable Restart Transaction error handling:

1. In the QALoad Conductor main window, select the **Script Assignment** tab.
2. In the **Error Handling** column, click the **Browse** Button and select **Restart Transaction** in the drop-down list.

Syntax

```
RESTART_TRANSACTION_TOP() ;
```

Parameters

None.

Example

```
BEGIN_TRANSACTION() ;
RESTART_TRANSACTION_TOP() ;
TRANSACTION CODE...
RESTART_TRANSACTION_BOTTOM() ;
DO_HttpCleanup() ;
DO_SomeOtherMiddlewareCleanup() ;
END_TRANSACTION() ;
```

Set

Applies to Visual Scripting. Assigns values to the Virtual Browser, Proxy, and other parts of the QALoad replay. This command sets the properties and attributes of the script.

 **Note:** For Visual Scripting, this command replaces the following EasyScript for WWW commands:

```
DO_AddHeader
DO_AttachFile
```

```

DO_BasicAuthorization
DO_Cache
DO_HttpVersion
DO_IPSpoofEnable
DO_NTLMAuthorization
DO_ProxyAuthorization
DO_ProxyEceptions
DO_SaveReplyType
DO_SetAssumedContentType
DO_SetBaudRate
DO_SetBaudRateEX
DO_SetJavascriptCleanupThreshold
DO_SetMaxBrowserThreads
DO_SetMaximumRetries
DO_SetRetryWait
DO_SetSSLConnectString
DO_SSLReuseSession
DO_SSLUseCipher
DO_SSLUseClientCert
DO_SSLUseProxy
DO_SetTimeout
DO_UsePersistentConnections
DO_UseProxy
    
```

Prototypes

```

boolean Set ( enumeration duration, enumeration bool_option, boolean boolean );
boolean Set ( enumeration duration, enumeration cache_option, enumeration cache_value );
boolean Set ( enumeration duration, enumeration int_option, integer integer );
boolean Set ( enumeration duration, enumeration proxy_option, enumeration proxy_mode_value );
boolean Set ( enumeration duration, enumeration string1_option, string string );
boolean Set ( enumeration duration, enumeration string2_option, string string1, string string2 );
boolean Set ( enumeration duration, enumeration string3_option, string string1, string string2, string string3 );
    
```

Return Value


Set returns true, if setting the option to the given value succeeded. Otherwise, Set returns false.

Parameters

Parameters	Description						
duration	<p>How long the new value lasts. Valid values include:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>EVERY_REQUEST</td> <td>Use this value for all following HTTP requests.</td> </tr> <tr> <td>NEXT_REQUEST</td> <td>Use this value for just the next HTTP request, then revert to the previous value.</td> </tr> </tbody> </table>	Value	Description	EVERY_REQUEST	Use this value for all following HTTP requests.	NEXT_REQUEST	Use this value for just the next HTTP request, then revert to the previous value.
Value	Description						
EVERY_REQUEST	Use this value for all following HTTP requests.						
NEXT_REQUEST	Use this value for just the next HTTP request, then revert to the previous value.						
bool_option	<p>Set boolean options. Valid values include:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>APPEND_CRLF_AFTER_POST_BODY</td> <td>Add " " after body (default is FALSE)</td> </tr> <tr> <td>IP_SPOOFING</td> <td>Turn on or off IP spoofing with the ipspool.dat datapool (default is FALSE)</td> </tr> </tbody> </table>	Value	Description	APPEND_CRLF_AFTER_POST_BODY	Add " " after body (default is FALSE)	IP_SPOOFING	Turn on or off IP spoofing with the ipspool.dat datapool (default is FALSE)
Value	Description						
APPEND_CRLF_AFTER_POST_BODY	Add " " after body (default is FALSE)						
IP_SPOOFING	Turn on or off IP spoofing with the ipspool.dat datapool (default is FALSE)						

	<p>JAVASCRIPT Turn Javascript handling on or off (default is TRUE)</p> <p>MANUALLY_SELECTED_SUBREQUESTS Skip all automatic sub-requests (default is FALSE)</p> <p>MINIMIZED_MEMORY_MODE Thin replay mode which prohibits Verify and Click_On (default is FALSE)</p> <p>REUSE_CONNECTION Try to keep connections alive between requests (default is TRUE)</p> <p>REUSE_SECURE_SESSION Save handshake information when possible (default is TRUE)</p>														
boolean	TRUE to enable the option; FALSE to disable the option														
cache_option	Set enumerated typed options. Valid value is: 1														
cache_value	<p>Set the virtual browser's caching system. Valid values include:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ALL</td> <td>Cache as much as possible (will degrade replay performance).</td> </tr> <tr> <td>AUTO</td> <td>Use the virtual browsers default settings.</td> </tr> <tr> <td>IMAGES</td> <td>Only images will be cached by the virtual browser.</td> </tr> <tr> <td>NO_CACHING</td> <td>No caching will be done by the virtual browser.</td> </tr> </tbody> </table>	Value	Description	ALL	Cache as much as possible (will degrade replay performance).	AUTO	Use the virtual browsers default settings.	IMAGES	Only images will be cached by the virtual browser.	NO_CACHING	No caching will be done by the virtual browser.				
Value	Description														
ALL	Cache as much as possible (will degrade replay performance).														
AUTO	Use the virtual browsers default settings.														
IMAGES	Only images will be cached by the virtual browser.														
NO_CACHING	No caching will be done by the virtual browser.														
int_option	<p>Set integer options. Valid values include:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>BROWSER_THREADS</td> <td>Maximum concurrent connections (default is 2).</td> </tr> <tr> <td>CONNECTION_RETRIES</td> <td>Maximum connection attempts (default is 4).</td> </tr> <tr> <td>RECEPTION_BAUD_RATE</td> <td>Network reception rate in bits per second.</td> </tr> <tr> <td>REFRESH_TIMEOUT</td> <td>Maximum meta refresh delay to wait for (default is 0).</td> </tr> <tr> <td>TRANSMISSION_BAUD_RATE</td> <td>Network transmission rate in bits per second.</td> </tr> <tr> <td>USER_PATIENCE</td> <td>Maximum seconds virtual user will wait (default is 120).</td> </tr> </tbody> </table>	Value	Description	BROWSER_THREADS	Maximum concurrent connections (default is 2).	CONNECTION_RETRIES	Maximum connection attempts (default is 4).	RECEPTION_BAUD_RATE	Network reception rate in bits per second.	REFRESH_TIMEOUT	Maximum meta refresh delay to wait for (default is 0).	TRANSMISSION_BAUD_RATE	Network transmission rate in bits per second.	USER_PATIENCE	Maximum seconds virtual user will wait (default is 120).
Value	Description														
BROWSER_THREADS	Maximum concurrent connections (default is 2).														
CONNECTION_RETRIES	Maximum connection attempts (default is 4).														
RECEPTION_BAUD_RATE	Network reception rate in bits per second.														
REFRESH_TIMEOUT	Maximum meta refresh delay to wait for (default is 0).														
TRANSMISSION_BAUD_RATE	Network transmission rate in bits per second.														
USER_PATIENCE	Maximum seconds virtual user will wait (default is 120).														
integer	A number that is the new value for the option.														
proxy_options	Set enumerated typed options. Valid value is: 1														
proxy_mode_value	Specify which technique will be used to set proxies. Valid values include:														

	Value	Description
	NO_PROXY	Direct connection to all servers.
	PROXY_AUTOMATIC_CONFIGURATION	PAC script determines proxying.
	MANUAL	Manually set proxies and proxy exceptions.
string1_option	Set string options (1 parameter). Valid values include:	
	Value	Description
	ADDITIONAL_SUBREQUEST	URL of a sub-request to add to the next request.
	BLOCK_TRAFFIC_FROM	Never make requests to matches in the list.
	BROWSER_IDENTITY	The identity string for User-Agent header field.
	CERTIFICATE	Set the browser's client certificate.
	CERTIFICATE_PASSWORD	Set a password for a client certificate.
	CGI_PARAMETER	CGI parameter to append to the URL of the request.
	CHECKPOINT_NAME	Name used for the next automatic checkpoint.
	CONNECT_REQUEST_FOR_SSL_TUNNELING	Set the connect message for an SSL proxy tunnel.
	DEFAULT_CONTENT_TYPE	The content type used if none is sent (default is "application/octet-stream").
	HTTP_VERSION	The version of HTTP to use (default is "1.1").
	ONLY_ALLOW_TRAFFIC_FROM	Only make requests to matches in the list.
	ONLY_USE_SSL_CIPHER	Set the SSL cipher to use for SSL requests.
	POST_DATA	Data to add to the post body in a name-value pair.
	PROHIBITED_CONTENT	Check the request for the provided criteria and report an error if the criteria are not found.
	PROXY_EXCEPTIONS	A list of host names that bypass the proxy.
	PROXY_HTTP_VERSION	The version of HTTP to use with the

	<p>proxy.</p> <p>PROXY_SCRIPT The URL of a Proxy Automatic Configuration script.</p> <p>REFERER Set the Referer header field for the next HTTP request.</p> <p>REQUIRED_CONTENT Check request for the provided criteria and report an error if the criteria are found.</p> <p>SIGNIFICANT_CONTENT_TYPE Content types to save replies for (default is "text/,application/x-javascript").</p> <p>SPOOFED_IP_ADDRESS Turn IP Spoofing on and set the browser's IP address.</p> <p>XML_DATA XML request document as a string.</p>																						
string	A string that contains the new value for the option.																						
string2_option	<p>Set string options (2 parameters). Valid values include:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>BASIC_AUTHORIZATION</td> <td>Basic authentication user name and password.</td> </tr> <tr> <td>CERTIFICATE</td> <td>Set the browser's client certificate and password.</td> </tr> <tr> <td>CGI_PARAMETER</td> <td>CGI Parameter's name and value for next Navigate_To or Post_To.</td> </tr> <tr> <td>COOKIE</td> <td>HTTP cookie name and value.</td> </tr> <tr> <td>HEADER</td> <td>Name and value for new header in requests.</td> </tr> <tr> <td>HTTP_PROXY</td> <td>Host and Port of HTTP proxy.</td> </tr> <tr> <td>PROXY_AUTHORIZATION</td> <td>Proxy authentication user name and password.</td> </tr> <tr> <td>POST_DATA</td> <td>POST data name and value for next Post_To.</td> </tr> <tr> <td>POST_FILE</td> <td>File to attach to next Post_To, local file name and full path.</td> </tr> <tr> <td>SECURE_PROXY</td> <td>Host and Port of a secure proxy.</td> </tr> </tbody> </table>	Value	Description	BASIC_AUTHORIZATION	Basic authentication user name and password.	CERTIFICATE	Set the browser's client certificate and password.	CGI_PARAMETER	CGI Parameter's name and value for next Navigate_To or Post_To.	COOKIE	HTTP cookie name and value.	HEADER	Name and value for new header in requests.	HTTP_PROXY	Host and Port of HTTP proxy.	PROXY_AUTHORIZATION	Proxy authentication user name and password.	POST_DATA	POST data name and value for next Post_To.	POST_FILE	File to attach to next Post_To, local file name and full path.	SECURE_PROXY	Host and Port of a secure proxy.
Value	Description																						
BASIC_AUTHORIZATION	Basic authentication user name and password.																						
CERTIFICATE	Set the browser's client certificate and password.																						
CGI_PARAMETER	CGI Parameter's name and value for next Navigate_To or Post_To.																						
COOKIE	HTTP cookie name and value.																						
HEADER	Name and value for new header in requests.																						
HTTP_PROXY	Host and Port of HTTP proxy.																						
PROXY_AUTHORIZATION	Proxy authentication user name and password.																						
POST_DATA	POST data name and value for next Post_To.																						
POST_FILE	File to attach to next Post_To, local file name and full path.																						
SECURE_PROXY	Host and Port of a secure proxy.																						
string1	A string that contains the new value for the option.																						
string2	A string that contains the new value for the option.																						
string3_option	<p>Set string options (3 parameters). Valid value is:</p> <p>NTLM_AUTHORIZATION: NTLM authentication domain, user name, and password.</p> <p> Note: NTLM user names and passwords can be variablized by machine, but not by user.</p>																						

string3

A string that contains the new value for the option.

Examples with virtual browser values

```
Set ( EVERY_REQUEST, BROWSER_IDENTITY, "Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)" );
Set ( EVERY_REQUEST, HTTP_VERSION, "1.1" );
Set ( EVERY_REQUEST, REUSE_CONNECTION, TRUE );
Set ( EVERY_REQUEST, JAVASCRIPT, TRUE );
Set ( EVERY_REQUEST, CERTIFICATE, GET_DATA_FIELD(CERTIFICATES, 1) );
Set ( EVERY_REQUEST, NTLM_AUTHORIZATION, "domain", "username", "password" );
Set ( EVERY_REQUEST, BASIC_AUTHORIZATION, "username", "password" );
Set ( EVERY_REQUEST, TRANSMISSION_BAUD_RATE, 9600 );
Set ( EVERY_REQUEST, RECEPTION_BAUD_RATE, 28800 );
Set ( EVERY_REQUEST, BROWSER_THREADS, 4 );
Set ( NEXT_REQUEST_ONLY, CGI_PARAMETER, "x", "1000" );
Set ( NEXT_REQUEST_ONLY, POST_DATA, "name", "jeff" );
Set ( NEXT_REQUEST_ONLY, POST_FILE, "file", "c:\\Data\\datafile.dat" );
```

Examples with miscellaneous values

```
Set ( EVERY_REQUEST, USER_PATIENCE, 120 );
Set ( EVERY_REQUEST, CONNECTION_RETRIES, 3 );
Set ( EVERY_REQUEST, SIGNIFICANT_CONTENT_TYPES, "text/*; application/x-javascript" );
Set ( EVERY_REQUEST, DEFAULT_CONTENT_TYPE, "text/html" );
Set ( NEXT_REQUEST_ONLY, COOKIE, "name=value; domain=.host.com; path=/" );
Set ( NEXT_REQUEST_ONLY, XMLREQUEST, "<?xml version='1.0'?>
  <element attribute='value'>text</element>" );
Set ( EVERY_REQUEST, HEADER, "name", "value" );
Set ( EVERY_REQUEST, CACHING, NONE );
Set ( EVERY_REQUEST, REUSE_SECURE_SESSION, TRUE );
Set ( EVERY_REQUEST, SPOOFED_IP_ADDRESS, GET_DATA_FIELD (IP_ADDRESSES, 1) );
Set ( EVERY_REQUEST, ALLOW_ONLY_TRAFFIC_FROM, "compuware.com, index.html" );
Set ( EVERY_REQUEST, BLOCK_TRAFFIC_FROM, "doubleclick.net, fastclick.net" );
```

Examples with proxy values

```
Set ( PROXY_MODE, PROXY_SCRIPT );
Set ( PROXY_SCRIPT, "http://172.22.222.23/" );
Set ( PROXY_MODE, MANUAL );
Set ( HTTP_PROXY, "proxyl.server.com", "80" );
Set ( SECURE_PROXY, "proxyl.server.com", "80" );
Set ( PROXY_EXCEPTIONS, "machine1.host.com, machine2, <local>" );
Set ( PROXY_HTTP_VERSION, "1.0" );
Set ( PROXY_AUTHORIZATION, "username", "password" );
```

Examples with advanced values

```
Set ( EVERY_REQUEST, ONLY_USE_SSL_CIPHER, "RC4-MD5" );
Set ( EVERY_REQUEST, CONNECT_REQUEST_FOR_SSL_TUNNELING,
  "CONNECT host:443 HTTP/1.1\r\n..." );
Set ( NEXT_REQUEST_ONLY, APPEND_CRLF_AFTER_POST_BODY, TRUE );
```

ShowMediaRP

Applies to Real Networks Streaming Media. Displays the media during a load test. Audio and video can be controlled separately. If video is enabled, a dialog box displays the video. For audio, the sound from the media will play through the sound device.

Notes:

- ! Exercise caution when using this feature. Use the audio display for one virtual user only. If enabling audio on two virtual users, audio from the two streams contends for the audio device. By default, audio and video do not display.

QALoad 5.02

Displaying the media for audio or video uses extra system resources and may degrade performance and skew test results.

- ! Real Networks streaming media is only supported in process mode. On the QALoad Player main window, in the Run As: group, select the Process option.
- ! Real Networks streaming media is only supported on a stand-alone QALoad Player or if the QALoad Player and QALoad Conductor are on the same machine.
- ! For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad" chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

```
ShowMediaRP( BOOL showAudio, BOOL showVideo );
```

Parameters

Parameter	Description
showAudio	Display and play audio.
showVideo	Display video.

Example

```
ShowMediaRP( FALSE, TRUE );  
// Display video, but leave audio muted
```

Verify

Applies to Visual Scripting. Used to verify expected text against an element of the page just requested.

Prototypes

```
boolean Verify ( enumeration type, string expected );  
boolean Verify ( enumeration type, enumeration specifier, string expected );
```

Return Value

Returns true if the text matches the text in the page. Otherwise, returns false.

Parameters

Parameter	Description
type	The type of verification to do. The types include: PAGE_TITLE: The HTML title of the page.
specifier	Options on how the matching is to be done. The specifier types include: ENTIRE: Match the entire title (default.) BEGINNING: Only match the beginning of the title. ENDING: Only match the end of the title.
expected	The text to verify against.

Examples

```
Verify ( PAGE_TITLE, "This is the page title" );  
Verify ( PAGE_TITLE, BEGINNING, "This is" );
```

WWW_FATAL_ERROR

Applies to HTTP and SSL requests. Also applies to Visual Scripting. WWW_FATAL_ERROR aborts or restarts a virtual user in the event of an error during replay.

This command handles error conditions in a script that invalidates the transaction. WWW_FATAL_ERROR is called internally by all script commands to report error conditions.

If Abort Transaction is selected in the Error Handling column of the QALoad Conductor Script Assignment tab, then WWW_FATAL_ERROR will abort the virtual user after generating a debug log and notifying the QALoad Conductor that it is aborting.

If Restart Transaction is selected in the Error Handling column, then WWW_FATAL_ERROR will restart the transaction from the restart point (DO_SetTransactionStart or RESTART_TRANSACTION_TOP) after generating a debug log and notifying the QALoad Conductor about the restart.

If Continue Transaction is selected in the Abort on Error Handling column, then the virtual user will continue as if no error had occurred. This may cause a virtual user middleware exception if WWW_FATAL_ERROR was called because the transaction is in an unstable state.

Syntax

```
WWW_FATAL_ERROR ( const char *short_desc, const char *long_desc ) ;
```

Parameters

Parameter	Description
short_desc	A string containing a one-word description of the error. This is often the name of the function where an error was encountered.
long_desc	A longer description of the error.

Example

```
WWW_FATAL_ERROR ( "My Func", "An error has occurred" ) ;
```

X_Coord

Applies to Visual Scripting. Marks the x_value parameter as an x-coordinate value.

Prototypes

```
X_Coord( string x_value );
```

Return Value

Returns the x-coordinate value.

Parameters

Parameter	Description
x_value	The x-coordinate value.

Example

```
// X_Coord and Y_Coord return the string passed into them. They are a
// label to make clicking on a server side imagemap easier to read.
```

QALoad 5.02

```
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/server-map.jpg", X_Coord("25"),  
Y_Coord("60"));  
  
// does the same as  
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/server-map.jpg", "25", "60");
```

XmlRequest

Applies to Visual Scripting. The XmlRequest function takes in the HTTP action and a URL and constructs a request to navigate to the URL.

If the method is "GET", XmlRequest makes a request for the URL expecting to get an XML reply. If the method is "POST", XmlRequest finishes an XML message and posts the message to the URL, expecting to get an XML reply.

XmlRequest is a direct replacement for Navigate_To and Post_To when the HTTP reply will contain XML.

Prototypes

```
boolean XMLRequest ( string method, string URL );
```

Return Value

Returns true, if the requested page is successfully retrieved. Otherwise, returns false.

Parameters

Parameter	Description
method	HTTP request method ("GET" or "POST")
URL	A URL containing the location of the page to be requested.

Examples

```
XmlRequest ( "GET", "http://msssoapsampleserver/  
MSSoapSamples/Echo/Service/Rpc/IsapiCpp/Echo.wsdl" );  
  
XmlRequest ( "POST",  
"http://MSSoapSampleServer:80/MSSoapSamples/Echo/Service/Rpc/IsapiCpp/Echo.wsdl" );
```

Y_Coord

Applies to Visual Scripting. Marks the y_value parameter as a y-coordinate value.

Prototypes

```
Y_Coord( string y_value );
```

Return Value

Returns the y-coordinate value.

Parameters

Parameter	Description
y_value	The y-coordinate value.

Example

```
// X_Coord and Y_Coord return the string passed into them. They are a
// label to make clicking on a server side imagemap easier to read.

Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/server-map.jpg", X_Coord("25"),
Y_Coord("60"));

// does the same as
Click_On(IMAGE, 1, SRC_ATTRIBUTE, "http://host.com/server-map.jpg", "25", "60");
```

Compuware customer support

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 QALoad installation. This information is displayed when you select the About command from any QALoad component's Help menu.
- ! Installation information, including installed components, whether it is installed in the default directories, and so on.
- ! 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 QALoad software, and the actions taken before the problem occurred.
- ! The exact QALoad error message, if any.
- ! The exact application, licensing, or operating system error messages, if any.
- ! Your Compuware client, office, or site number, if available.

Contact information

Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

(800) 538-7822

World Wide Web Information

To access Compuware Corporation's site on the 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 fast access to critical information about your QACenter product. You can read or download documentation, frequently asked questions, and product fixes, or email your questions or comments. To access FrontLine, you must first register and obtain a password. To register, point your browser at <http://frontline.compuware.com>.

QALoad glossary

Non-alphabetic

2-tier

3-tier

A

ActiveData

Analyze

B

batch test

C

capture: see [recording](#)

.CAP file: see [capture file](#)

capture file

checkpoints

checkpoint duration

[concurrent users](#)

Conductor

[conversion](#)

convert: see [conversion](#)

[counters](#)

D

datapool

E

[EasyScript session](#)

error handling

F

[Function Wizard](#)

G

H

I

ICA files
IP spoofing

J

K

L

load testing

M

metrics
middleware
middleware session: see [session \(middleware\)](#)

N

n-tier

O

P

pacing
parameterization: see [variablization](#)
performance testing
Player
Player agent
process mode
.PTF file

Q

R

ramp-up script
.REC file
recording
Remote Monitoring
replay

response time
.RIP file

S

Script Development Workbench
service level threshold
session (middleware)
session ID file
sleep factor
sleep
source variable
stress testing
synchronization

T

thread mode
thresholds
.TIM file
timing files
traffic
transaction
transaction cleanup
transaction duration
transaction loop
transaction pacing: [see pacing](#)
transaction throughput

U

Universal session

V

validation
variablization
virtual user
Visual Navigator
visual scripting

W

Web User Monitor

X

Y

Z

QALoad 5.02

Test Execution

Hits (as in, hits per second)

Hits per Second

.CSV File

Rectangular Datapool

QARun

Compiler

Compiler Options

Analyze

Analysis

Conductor

Player

Player Agent

Windows

Unix

Linux

'C'

'C++'

Programming Language

Methodology

Service Level Threshold

Session Duration

Load Test Recording

Debugging

Debug Options

DO_SLEEP

Timing Options

External Data

Binary Files

ASCII Files

Server Analysis Agent

SNMP

Application Expert

Application Vantage

Server Vantage

VU

Virtual User Increment

Time Interval
analyze (QALoad)
application under test (AUT)
asynchronous
automated (v. manual)
capture
checkpoint
client
compile
conductor (QALoad)
convert
error
function
KPI-key performance indicator
manual (v. automated)
metrics
reponse time
request
response
script
server
session (.id file)
stateless
synchronous
test plan
testing methodology
workbench (QALoad)

Accessibility features

Compuware is committed to compliance with Section 508 standards for accessibility in software products. Section 508 standards were enacted by Congress in 1998 as an amendment to the Rehabilitation Act. The standards require federal agencies to increase the accessibility of their electronic information to people with disabilities. Software that is fully compliant with Section 508 standards provides information access to people with disabilities that is comparable to the information access available to people without disabilities. Software that is compliant with exceptions, such as QALoad, provides support for most, but not all, criteria of compliance.

This topic describes the level of accessibility currently available in QALoad, including the product documentation.

Accessibility of QALoad components

All components of QALoad — Script Development Workbench, Conductor, Player, and Analyze — support the basic criteria of accessibility for software applications. For most categories of compliance, there are exceptions in which assistive technologies do not work in all situations or the alternate methods of information retrieval or product function are not always available. However, the following basic accessibility features are available in QALoad:

- ! Product functions are executable from the keyboard
- ! The application does not disrupt or disable activated features of other products that are identified as accessibility features
- ! The product provides an on-screen indication of the current focus and the focus can be tracked by assistive technologies
- ! The identity, operation, and state of elements of the user interface are available to assistive technologies
- ! Textual information is provided through the operating system functions for displaying text
- ! The application does not override user-selected contrast and color selections
- ! Color-coding is not used as the only means of conveying information
- ! Electronic forms allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form

Accessibility of QALoad documentation

Generally, the QALoad documentation is accessible and can be read by screen readers. Alternate formats of the documentation are not currently available.

Online help

QALoad's online help system is HTML-based and can be read by screen readers. All graphics in the online help have text descriptions. Topics in the online help can be printed.

Release Notes

The QALoad Release Notes is an HTML document that can be read by screen readers and printed.

Installation Guide

The QACenter Performance Edition Installation and Configuration Guide is an accessible PDF-based book that provides text descriptions for images in addition to the ability to be read by some screen readers and the

Read Aloud feature of Adobe Acrobat. To take advantage of the accessibility features in the Installation Guide, you must have Adobe Acrobat Reader 6.0 or later. This book can be printed.

Assistive technology tools that enhance the accessibility of QALoad

There are many third-party assistive technology tools that you can use to access the QALoad user interface and product documentation by alternate means. These products perform a variety of functions, such as enlarging the user interface and reading text on the interface or in the documentation. Many features are available in Windows, such as:

- ! Windows Narrator
- ! Windows accessibility wizard (includes features such as SoundSentry, ShowSounds, StickyKeys, and MouseKeys)
- ! Windows On-Screen Keyboard
- ! Microsoft Magnifier

Refer to the Windows documentation to learn more about using these features. Most other tools fall into one of the following two categories:

- ! **Screen readers:** Software programs that present graphics and text as speech. A screen reader is used to verbalize, or "speak," everything on the screen including names and descriptions of control buttons, menus, text, and punctuation. (Example: JAWS for Windows by Freedom Scientific, Inc.)
- ! **Screen enlargers (or screen magnifiers):** Software that works like a magnifying glass on other applications. They enlarge a portion of the screen as the user moves the focus — increasing legibility for some users. Some screen enlargers allow you to zoom in and out on a particular area of the screen. (Example: ZoomText Magnifier by AI Squared)

Product shortcut keys

The following tables list the defined shortcut keys for each component of QALoad. You can use these key combinations to open dialog boxes, start or stop processes, or interact in other ways with QALoad without using a mouse.

Script Development Workbench

Shortcut Key Combination	Action
CTRL+N	Open the New dialog box, from which you can create new scripts or datapools.
CTRL+O	Open the Open dialog box, from which you can open various product files.
CTRL+S	Save the active document.
CTRL+P	Print the active document
CTRL+Z	Undo the last action.
CTRL+Y	Redo the last action.
CTRL+X	Cut.
CTRL+C	Copy.

CTRL+V	Paste.
CTRL+F	Find.
CTRL+H	Replace.
CTRL+G	Open the Go to Line dialog box, from which you can go to a specific line number in the active script.
CTRL+F2	Toggle bookmark on/off.
F2	Move to next bookmark.
SHIFT+F2	Move to previous bookmark.
CTRL+SHIFT+F2	Clear all bookmarks.
CTRL+ALT+C	Start a recording.
CTRL+ALT+T	Stop a recording.
CTRL+ALT+R	Restart a recording.
CTRL+ALT+U	Resume a recording.
CTRL+ALT+A	Animate a recording.
CTRL+ALT+S	Step to the next response in an animation.
CTRL+ALT+B	Step back to the previous response in an animation.
F7	Compile the active script.
CTRL+F5	Validate the active script.
CTRL+T	Open the FTP Transfer dialog box, from which you can send a file via FTP.
F1	Open the QALoad online help.
ALT+F4	Close the Script Development Workbench.

Conductor

Shortcut Key Combination	Description
F1	Open the QALoad online help.
CTRL+X	Cut.
CTRL+C	Copy.
CTRL+P	Paste.

ALT+F4	Close the Conductor.
--------	----------------------

Analyze

Shortcut Key Combination	Description
CTRL+O	Open a timing file.
CTRL+P	Print the active document.
CTRL+C	Copy Summary and Data selections.
CTRL+A	Select all items.
CTRL+U	Unselect all items.
F1	Open QALoad online help.
ALT+F6	Move to the next tab in the Data window.
ALT+F4	Close Analyze.

Player

Shortcut Key Combination	Description
CTRL+N	Clear any current messages from the Player's main window.
CTRL+S	Save a text file of the messages reported by a QALoad Player during a test.
CTRL+P	Print Player output.
CTRL+Z	Undo the last action.
CTRL+X	Cut.
CTRL+C	Copy.
CTRL+V	Paste.
F1	Open QALoad online help.
ALT+F4	Close the Player.

To request a Voluntary Product Accessibility Template (VPAT) or for more information regarding accessibility of QALoad or any other Compuware product, see <http://www.compuware.com/accessibility/>

Compuware customer support

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 QALoad installation. This information is displayed when you select the About command from any QALoad component's Help menu.
- ! Installation information, including installed components, whether it is installed in the default directories, and so on.
- ! 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 QALoad software, and the actions taken before the problem occurred.
- ! The exact QALoad error message, if any.
- ! The exact application, licensing, or operating system error messages, if any.
- ! Your Compuware client, office, or site number, if available.

Contact information

Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

(800) 538-7822

World Wide Web Information

To access Compuware Corporation's site on the 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 fast access to critical information about your QACenter product. You can read or download documentation, frequently asked questions, and product fixes, or email your questions or comments. To access FrontLine, you must first register and obtain a password. To register, point your browser at <http://frontline.compuware.com>.

Index

.	
.cap file.....	64, 155
.cfg file.....	207
.cpp.....	155
.log file	64
.REC file	
Conductor recording overview	219
.rfd.....	155
.rip file	
Logfile Generation	211
.rip file.....	64
.VisHtml	155
.VisTree.....	155
.VisXml	155
.zip file, creating in Analyze	
Creating a zip file of test results	268
–	
_xClone.....	351
_xResync.....	351
_xSave.....	352
A	
accessing test data	250
Action item	
Page sub-item	143
Action item	149
ActiveData for Oracle	
using the Compare Tool	83
Add	534
adding a variable.....	162
adding Players to a test	208
adding virtual users at runtime.....	202
Additional SubRequests	143
AddListBoxValue.....	534
AddNew	352
AddrByte	767
ADO	
recording options	66
ADO	66
ADO	271
ADO method reference.....	66
ADO_Command	285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296
ADO_Command(n)->Cancel	285
ADO_Command(n)->CreateParameter	286
ADO_Command(n)->Execute.....	287
ADO_Command(n)->GetCommandStream	287
ADO_Command(n)->GetCommandText	288
ADO_Command(n)->GetCommandTimeout..	288
ADO_Command(n)->GetCommandType	289
ADO_Command(n)->GetDialect	289
ADO_Command(n)->GetName	290
ADO_Command(n)->GetNamedParameters ...	290
ADO_Command(n)->GetParameters.....	290
ADO_Command(n)->GetPrepared	291
ADO_Command(n)->GetProperties.....	291
ADO_Command(n)->PutActiveConnection	292
ADO_Command(n)->PutCommandText	292
ADO_Command(n)->PutCommandTimeout ..	293
ADO_Command(n)->PutCommandType.....	293
ADO_Command(n)->PutDialect.....	294
ADO_Command(n)->PutName	294
ADO_Command(n)->PutNamedParameters....	295
ADO_Command(n)->PutPrepared	295
ADO_Command(n)->PutRefActiveConnection	296
ADO_Connect	296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308

ADO_Connect(n)->GetIsolationLevel.....	301	ADO_Field(n)->GetOriginalValue.....	313
ADO_Connect(n)->BeginTrans.....	296	ADO_Field(n)->GetPrecision.....	314
ADO_Connect(n)->Close.....	297	ADO_Field(n)->GetProperties.....	314
ADO_Connect(n)->CommitTrans.....	297	ADO_Field(n)->GetStatus.....	315
ADO_Connect(n)->Execute.....	298	ADO_Field(n)->GetType.....	315
ADO_Connect(n)->GetAttributes.....	298	ADO_Field(n)->GetUnderlyingValue.....	316
ADO_Connect(n)->GetCommandTimeout.....	299	ADO_Field(n)->GetValue.....	316
ADO_Connect(n)->GetConnectionString.....	299	ADO_Field(n)->PutAttributes.....	316
ADO_Connect(n)->GetConnectionTimeout.....	300	ADO_Field(n)->PutDataFormat.....	317
ADO_Connect(n)->GetCursorLocation.....	300	ADO_Field(n)->PutDefinedSize.....	317
ADO_Connect(n)->GetDefaultDatabase.....	300	ADO_Field(n)->PutNumericScale.....	318
ADO_Connect(n)->GetMode.....	301	ADO_Field(n)->PutPrecision.....	318
ADO_Connect(n)->GetProvider.....	302	ADO_Field(n)->PutType.....	319
ADO_Connect(n)->GetState.....	302	ADO_Field(n)->PutValue.....	319
ADO_Connect(n)->GetVersion.....	303	ADO_FieldSet.....	320, 321, 322, 323, 324
ADO_Connect(n)->Open.....	303	ADO_FieldSet(0)->GetNewEnum.....	320
ADO_Connect(n)->OpenSchema.....	304	ADO_FieldSet(n)->Append.....	320
ADO_Connect(n)->PutAttributes.....	304	ADO_FieldSet(n)->Append15.....	321
ADO_Connect(n)->PutCommandTimeout.....	305	ADO_FieldSet(n)->CancelUpdate.....	321
ADO_Connect(n)->PutConnectionString.....	305	ADO_FieldSet(n)->Delete.....	322
ADO_Connect(n)->PutConnectionTimeout.....	306	ADO_FieldSet(n)->GetCount.....	322
ADO_Connect(n)->PutCursorLocation.....	306	ADO_FieldSet(n)->GetItem.....	323
ADO_Connect(n)->PutDefaultDatabase.....	306	ADO_FieldSet(n)->Refresh.....	323
ADO_Connect(n)->PutIsolationLevel.....	307	ADO_FieldSet(n)->Resync.....	324
ADO_Connect(n)->PutMode.....	307	ADO_FieldSet(n)->Update.....	324
ADO_Connect(n)->PutProvider.....	308	ADO_IEnum.....	324
ADO_Connect(n)->RollbackTrans.....	308	ADO_IEnum(n)->NextProperty.....	324
ADO_Field309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319		ADO_IEnumField.....	325
ADO_Field(n)->AppendChunk.....	309	ADO_IEnumField(n)->NextField.....	325
ADO_Field(n)->GetActualSize.....	310	ADO_IEnumParameter.....	326
ADO_Field(n)->GetAttributes.....	310	ADO_IEnumParameter(n)->NextParameter.....	326
ADO_Field(n)->GetChunk.....	311	ADO_LoadVariant.....	326
ADO_Field(n)->GetDataFormat.....	312	ADO_LoadVariant(n).....	326
ADO_Field(n)->GetDefinedSize.....	312	ADO_Parameter327, 328, 329, 330, 331, 332, 333, 334	
ADO_Field(n)->GetName.....	312	ADO_Parameter(n)->AppendChunk.....	327
ADO_Field(n)->GetNumericScale.....	313	ADO_Parameter(n)->GetAttributes.....	327

ADO_Parameter(n)->GetDirection	328	ADO_Record(n)->DeleteRecord	343
ADO_Parameter(n)->GetName	328	ADO_Record(n)->GetActiveConnection	344
ADO_Parameter(n)->GetNumericScale	329	ADO_Record(n)->GetChildren	344
ADO_Parameter(n)->GetPrecision	329	ADO_Record(n)->GetFields.....	345
ADO_Parameter(n)->GetSize.....	330	ADO_Record(n)->GetMode.....	345
ADO_Parameter(n)->GetValue.....	330	ADO_Record(n)->GetParentURL.....	346
ADO_Parameter(n)->PutAttributes	331	ADO_Record(n)->GetRecordType.....	346
ADO_Parameter(n)->PutDirection	331	ADO_Record(n)->GetSource	347
ADO_Parameter(n)->PutName.....	332	ADO_Record(n)->GetState.....	347
ADO_Parameter(n)->PutNumericScale	332	ADO_Record(n)->MoveRecord	347
ADO_Parameter(n)->PutPrecision	333	ADO_Record(n)->Open	348
ADO_Parameter(n)->PutSize	333	ADO_Record(n)->PutActiveConnection	349
ADO_Parameter(n)->PutType	334	ADO_Record(n)->PutMode.....	350
ADO_Parameter(n)->PutValue	334	ADO_Record(n)->PutRefActiveConnection	350
ADO_ParameterSet	335, 336, 337	ADO_Record(n)->PutSource.....	350
ADO_ParameterSet(n)->Append	335	ADO_Recordset 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387	
ADO_ParameterSet(n)->Delete.....	335	ADO_Recordset(n)->_xClone.....	351
ADO_ParameterSet(n)->GetCount	336	ADO_Recordset(n)->_xResync	351
ADO_ParameterSet(n)->GetItem	336	ADO_Recordset(n)->_xSave	352
ADO_ParameterSet(n)->GetNewEnum	337	ADO_Recordset(n)->AddNew	352
ADO_ParameterSet(n)->Refresh	337	ADO_Recordset(n)->Cancel	353
ADO_Property	337, 338, 339, 340	ADO_Recordset(n)->CancelBatch	353
ADO_Property(n)->GetAttributes	337	ADO_Recordset(n)->CancelUpdate	353
ADO_Property(n)->GetName.....	338	ADO_Recordset(n)->Clone	354
ADO_Property(n)->GetType.....	338	ADO_Recordset(n)->Close	354
ADO_Property(n)->GetValue	339	ADO_Recordset(n)->CompareBookmarks	355
ADO_Property(n)->PutValue.....	340	ADO_Recordset(n)->Delete.....	356
ADO_PropertySet	340, 341, 342	ADO_Recordset(n)->Find	356
ADO_PropertySet(n)->GetCount	340	ADO_Recordset(n)->GetAbsolutePage.....	357
ADO_PropertySet(n)->GetItem	341	ADO_Recordset(n)->GetAbsolutePosition	357
ADO_PropertySet(n)->GetNewEnum	341	ADO_Recordset(n)->GetActiveCommand	358
ADO_PropertySet(n)->Refresh.....	342	ADO_Recordset(n)->GetActiveConnection	358
ADO_Record....	342, 343, 344, 345, 346, 347, 348, 349, 350	ADO_Recordset(n)->GetBOF.....	359
ADO_Record(n)->Cancel	342	ADO_Recordset(n)->GetBookmark	359
ADO_Record(n)->Close.....	342		
ADO_Record(n)->CopyRecord	343		

ADO_Recordset(n)->GetCacheSize	359	ADO_Recordset(n)->PutCollect	377
ADO_Recordset(n)->GetCollect	360	ADO_Recordset(n)->PutCursorLocation	377
ADO_Recordset(n)->GetCursorLocation	360	ADO_Recordset(n)->PutCursorType	378
ADO_Recordset(n)->GetCursorType	361	ADO_Recordset(n)->PutDataMember	378
ADO_Recordset(n)->GetDataMember	361	ADO_Recordset(n)->PutFilter	379
ADO_Recordset(n)->GetDataSource	362	ADO_Recordset(n)->PutIndex	379
ADO_Recordset(n)->GetEditMode	362	ADO_Recordset(n)->PutLockType	379
ADO_Recordset(n)->GetEOF	363	ADO_Recordset(n)->PutMarshalOptions.....	380
ADO_Recordset(n)->GetFields	363	ADO_Recordset(n)->PutMaxRecords	380
ADO_Recordset(n)->GetFilter	364	ADO_Recordset(n)->PutPageSize	381
ADO_Recordset(n)->GetIndex	364	ADO_Recordset(n)->PutRefActiveConnection	381
ADO_Recordset(n)->GetLockType	364	ADO_Recordset(n)->PutRefDataSource	382
ADO_Recordset(n)->GetMarshalOptions.....	365	ADO_Recordset(n)->PutRefSource.....	382
ADO_Recordset(n)->GetMaxRecords.....	365	ADO_Recordset(n)->PutSort	383
ADO_Recordset(n)->GetPageCount	366	ADO_Recordset(n)->PutSource	383
ADO_Recordset(n)->GetPageSize	366	ADO_Recordset(n)->PutStayInSync.....	383
ADO_Recordset(n)->GetProperties.....	366	ADO_Recordset(n)->ReQuery	384
ADO_Recordset(n)->GetRecordCount	367	ADO_Recordset(n)->Resync.....	384
ADO_Recordset(n)->GetRows	367	ADO_Recordset(n)->Save	385
ADO_Recordset(n)->GetSort	368	ADO_Recordset(n)->Seek	386
ADO_Recordset(n)->GetSource.....	368	ADO_Recordset(n)->Supports.....	386
ADO_Recordset(n)->GetState.....	369	ADO_Recordset(n)->Update.....	386
ADO_Recordset(n)->GetStatus.....	369	ADO_Recordset(n)->UpdateBatch	387
ADO_Recordset(n)->GetStayInSync.....	370	ADO_Stream ...	388, 389, 390, 391, 392, 393, 395, 396, 397, 398, 399, 400
ADO_Recordset(n)->GetString.....	370	ADO_Stream(n)->Cancel	388
ADO_Recordset(n)->Move	371	ADO_Stream(n)->Close.....	388
ADO_Recordset(n)->MoveFirst	371	ADO_Stream(n)->CopyTo.....	388
ADO_Recordset(n)->MoveLast	372	ADO_Stream(n)->Flush	389
ADO_Recordset(n)->MoveNext.....	372	ADO_Stream(n)->GetCharset	390
ADO_Recordset(n)->MovePrevious.....	373	ADO_Stream(n)->GetEOS.....	390
ADO_Recordset(n)->NextRecordset	373	ADO_Stream(n)->GetLineSeparator.....	391
ADO_Recordset(n)->Open	374	ADO_Stream(n)->GetMode.....	391
ADO_Recordset(n)->PutAbsolutePage.....	374	ADO_Stream(n)->GetPosition.....	391
ADO_Recordset(n)->PutAbsolutePosition.....	375	ADO_Stream(n)->GetSize.....	392
ADO_Recordset(n)->PutActiveConnection.....	375	ADO_Stream(n)->GetState.....	392
ADO_Recordset(n)->PutBookmark.....	376	ADO_Stream(n)->GetType.....	393
ADO_Recordset(n)->PutCacheSize.....	376		

ADO_Stream(n)->LoadFromFile.....	393	basic, in Visual Navigator scripts.....	149
ADO_Stream(n)->PutCharset	395	NTLM, in Visual Navigator scripts	149
ADO_Stream(n)->PutLineSeparator	395	B	
ADO_Stream(n)->PutMode.....	396	batch test	
ADO_Stream(n)->PutPosition	396	adding sessions.....	208
ADO_Stream(n)->PutType.....	396	running.....	6
ADO_Stream(n)->Read	397	terminating.....	216
ADO_Stream(n)->ReadText	397	BEGIN_TRANSACTION.....	622
ADO_Stream(n)->SaveToFile.....	398	BEGIN_UENTITY	702
ADO_Stream(n)->SetEOS.....	398	BeginBlock	403
ADO_Stream(n)->SkipLine.....	399	BeginCheckpoint	622
ADO_Stream(n)->Write.....	399	BeginTrans	296
ADO_Stream(n)->WriteText.....	400	BlockScroller	536
ADOSream	394	bulk license checkout	213
ADOSream(n)->Open.....	394	Button	536
AlertDialog	535	C	
Analyze		Cancel	285, 342, 353, 388
buttons.....	246	CancelBatch	353
menus	246	CancelQueryDialog.....	537
opening.....	250	CancelUpdate	321, 353
overview	245	capture file	
test statistics.....	221	insert commands.....	16
toolbars	246	capture file.....	16
Analyze.....	245	Certificate Authority.....	181
Analyze.....	245	CfmOLE	538
Analyze.....	246	CfmVBX	538
Analyzing Load Test Data	220	CGI parameters	
Append.....	320, 335	in Visual Navigator scripts.....	149
Append15.....	321	CGI parameters.....	149
AppendChunk.....	309, 327	character encoding	797
Application Expert	201, 228, 229, 230, 266	check out/in licenses	213
Application Vantage	201, 228, 229, 230, 266	Checkbox	539
AppTimer	535	CheckForErrorMsgs.....	539
assigning machines to a test	197	checkpoint duration	
assigning scripts to Players.....	193	Top Ten Longest Checkpoint Durations report	
Attach	798	263
authentication		understanding durations.....	245
		checkpoint pair, Visual Navigator script item .	141

checkpoints		
detail	251	
duration	245	
graphing.....	217, 255	
verifying.....	65	
Citrix		
conversion options.....	69	
dynamic windows.....	22, 23, 70, 71	
ICA files.....	69	
overview	66	
recording options.....	68	
server farms.....	24, 66, 73	
troubleshooting	190	
Citrix	66	
Citrix command index.....	401	
CitrixInit	404	
CitrixUninit	404	
Clear	799	
Click On Button (submit)	143, 148	
Click On Link.....	143, 147	
Click, Citrix command	405	
Click_On	801	
client certificate		
in a datapool	4	
passwords.....	140	
client certificate.....	140	
client certificate.....	181	
client certificate.....	231	
Client Throughput report	262	
Clone.....	354	
Close.....	297, 342, 354, 388	
CLOSE_ALL_DATA_POOLS.....	623	
CLOSE_DATA_POOL.....	623	
command		
DO_WSK_Send.....	117	
insert into capture file	16	
command	16	
command line		
starting Conductor	214	
command line.....	214	
command line.....	243	
comment, Visual Navigator script item	142	
CommitTrans.....	297	
Compare Tool		
ActiveData for Oracle	83	
CompareBookmarks.....	355	
concurrent license	213	
concurrent users		
Concurrent Users report	260	
Conductor		
interface.....	191	
menus and toolbar buttons.....	200	
starting the Conductor	248	
Conductor.....	200	
configuring		
Application Expert test	230	
Application Vantage test	230	
browser	167	
Conductor.....	203	
test machines.....	194	
Connect, Citrix command.....	406	
Connect, Oracle Forms Server command	540	
Content Check.....	143	
conversion options		
Citrix.....	69	
Oracle.....	80	
Oracle Forms Server	87	
SAP.....	102, 106	
TUXEDO	112	
UNIFACE.....	116	
Winsock	121	
WWW	169	
conversion options.....	80	
conversion options.....	87	

conversion options	112	CtxSetEnableCounters.....	412
conversion options	116	CtxSetEnableWildcardMatching	413
conversion options	169	CtxSetICAFile.....	413
cookies		CtxSetLoginInfo.....	413
in Visual Navigator scripts.....	145, 149	CtxSetPingTimeout	414
cookies.....	60	CtxSetWaitPointTimeout	414
cookies.....	177	CtxSetWindowMatchTitle.....	414
Cookies Set by Server	143	CtxSetWindowRetries.....	415
CopyRecord.....	343	CtxSetWindowTimeout	415
CopyTo.....	388	CtxSetWindowVerification.....	415
COUNTER_VALUE.....	623	CtxType.....	416
counters		CtxTypeChar.....	416
custom	17	CtxTypeVK.....	417
Detail data.....	252	CtxWaitForCaptionChange.....	417
graphing.....	256	CtxWaitForScreenUpdate	417
Remote Monitoring	222	CtxWaitForWindowActivate	418
counters.....	252	CtxWaitForWindowCreate	418
CPU usage	201	CtxWaitforWindowDestroy.....	419
CreateParameter	286	CtxWaitForWindowLglconChange.....	419
CSV reports	268	CtxWaitForWindowMinimize.....	419
CTX_error_handler	404	CtxWaitForWindowMove	420
CtxClick	405	CtxWaitForWindowResize.....	420
CtxConnect.....	406	CtxWaitForWindowSmIconChange.....	421
CtxDisconnect	406	CtxWaitforWindowStyleChange.....	421
CtxDoubleClick.....	406	CtxWindowEventExists.....	25, 73, 422
CtxKeyDown	407	custom counters	17, 252
CtxKeyUp	407	custom error handling.....	193, 203
CtxMouseDown	408	custom script messages.....	17
CtxMouseMove.....	408	customer support	864, 874
CtxMouseUp	409	D	
CtxPing	409	data thinning.....	203, 251
CtxPoint	410	datapoints	
CtxScreenEventExists.....	25, 73, 410	graphing	255
CtxSetApplication	410	thinning.....	255
CtxSetCitrixPort	411	datapoints.....	255
CtxSetConnectTimeout	411	datapool variable	
CtxSetDisconnectTimeout	411	renaming	162

datapools	
client certificate	4
creating	19, 160
importing.....	160
incorporating.....	19
inserting in a script	157, 161
modifying	19, 160
NetLoad.....	184, 185
removing used data	210, 216
retrieving.....	19
substituting a string.....	50, 126, 158, 645
Visual Navigator.....	160
datapools.....	184
datapools.....	210
datapools.....	216
Datapools and Variables dialog box	160
DATE_TIME.....	624
DB2	
DB2/UNIX playback	186, 243
DB2.....	423
DBCS	
Visual Navigator.....	133
visual scripting.....	133, 168
DBCS.....	133
DBCS.....	168
debug print, Visual Navigator script item	142
debug trace.....	211
debugging	
logfiles.....	64
debugging.....	211
decrement variable, Visual Navigator script item	142
DefaultCheckpointsOn	625
define transaction loop.....	18
DEFINE_COUNTER.....	626
DEFINE_TRANS_TYPE.....	626
Delete	322, 335, 356
DeleteRecord.....	343
Detail data	
checkpoints.....	251
counters.....	252
Top Processes.....	254
Detail data.....	251
Detail data.....	254
Details view, Conductor	198
dial-up virtual users.....	202
directory options	12
DisableStatisticsRP	803
Disconnect	540
Disconnect, Citrix command	406
DisplayErrorDialog	541
DisplayList	541
DO_AbortOnError.....	627
DO_AddHeader	803
DO_AdditionalSubRequest	833
DO_AllowTrafficFrom	804
DO_AttachFile.....	805
DO_autocommitoff.....	475
DO_autocommiton	476
DO_AutomaticSubRequests.....	805
DO_BasicAuthorization	807
DO_binddate.....	476
DO_BindForUpdateRowID	477
DO_bindnull	477
DO_bindstring	478
DO_BindV	479
DO_BlankOutOfRangeData	808
DO_BlockTrafficFrom	808
DO_Cache.....	809
DO_cleanup	480
DO_Clear	809
DO_ClearCache	811
DO_ClearDNSCache	811
DO_ClearJavascript	812

DO_commit.....	480	DO_init_indp.....	485
DO_DynamicCookieHandling.....	812	DO_InitAWL.....	645
DO_DynamicRedirectHandling.....	814	DO_initDB2.....	425
DO_EnableJavascript.....	815	DO_InitHttp.....	831
DO_EncodeString.....	815	DO_initODBC.....	436
Do_ExtractString.....	628	DO_IPSpooferEnable.....	832
DO_FetchIters.....	480	DO_LoadMem.....	442
DO_free_data.....	492	DO_Logfile_PSV.....	724
DO_FreeDB2.....	424	DO_Logfile_URB.....	703
DO_FreeHttp.....	816	DO_makedate.....	493
DO_freeitem.....	492	DO_MSLEEP.....	629
DO_FreeODBC.....	435	DO_NTLMAuthorization.....	832
DO_get_select_variable.....	481	DO_OCI8BindDate.....	501
DO_GetAnchorByNumber.....	816	DO_OCI8BindNull.....	501
DO_GetAnchorCount.....	817	DO_OCI8BindString.....	502
DO_GetAnchorHREF.....	817	DO_OCI8GetSelectData.....	502
DO_GetAnchorHREFEx.....	819	DO_OCI8InitAlen.....	504
DO_GetAnchorHREFn.....	820	DO_OCI8InitIndp.....	504
DO_GetClientMapHREF.....	821	DO_OCIAttrSet.....	505
DO_GetCookie.....	822	DO_OCIBind.....	506
DO_GetCookieFromReplyEx.....	823	DO_OCICommit.....	507
DO_GetFormActionStatement.....	824	DO_OCIDefine.....	508
DO_GetFormValueByName.....	825	DO_OCIDescriptorAlloc.....	509
DO_GetHeaderFromReply.....	826	DO_OCIDescriptorFree.....	509
DO_GetLastHttpError.....	826	DO_OCIEnvFreeAll.....	509
DO_GetOutputData.....	492	DO_OCIEnvInit.....	510
DO_GetRedirectedURL.....	827	DO_OCIExecute.....	510
DO_GetReplyBuffer.....	828	DO_OCIHandleAlloc.....	511
DO_GetSelectData.....	482	DO_OCIHandleFree.....	511
DO_GetUniqueString.....	828	DO_OCIInitialize.....	512
DO_GetUniqueStringEx.....	829	DO_OCILdaToSvcCtx.....	512
DO_Http.....	830	DO_OCILOBRead.....	513
DO_HttpCleanup.....	830	DO_OCILOBWrite.....	514
DO_Https.....	672	DO_OCILogoff.....	515
DO_HttpVersion.....	831	DO_OCILogoffEx.....	515
DO_init_alen.....	483	DO_OCILogon.....	515
DO_init_data.....	484	DO_OCIProcessSelectList.....	516

DO_OCIProcessSelectList_EX.....	517	DO_PSV_logon.....	739
DO_OCIRollback.....	518	DO_PSV_long2uf.....	740
DO_OCIServerAttach.....	518	DO_PSV_modify.....	741
DO_OCIServerDetach.....	519	DO_PSV_ndelete.....	742
DO_OCISessionBegin.....	520	DO_PSV_open.....	743
DO_OCISessionEnd.....	520	DO_PSV_remotepath.....	745
DO_OCISmtExecute.....	521	DO_PSV_rollback.....	746
DO_OCISmtPrepare.....	522	DO_PSV_select.....	746
DO_OCISmtPrepare_EX.....	522	DO_PSV_selectdb.....	749
DO_OCISvcCtxToLda.....	523	DO_PSV_setocc.....	750
DO_OCITransCommit.....	524	DO_PSV_sql.....	751
DO_OCITransRollback.....	524	DO_PSV_sselect.....	752
DO_oclose.....	485	DO_PSV_str2uf.....	754
DO_oexec.....	486	DO_PSV_uf2bin.....	755
DO_olog.....	486	DO_PSV_uf2dbl.....	756
DO_ologof.....	487	DO_PSV_uf2long.....	756
DO_oopen.....	487	DO_PSV_uf2str.....	757
DO_oopt.....	487	DO_PSV_update.....	758
DO_oparse.....	488	DO_PSV_write.....	760
DO_process_select_list.....	488	DO_PSV_xtrans.....	761
DO_ProxyAuthorization.....	834	DO_PSV_zero.....	762
DO_ProxyExceptions.....	834	DO_rollback.....	489
DO_PSV_bin2uf.....	724	Do_SAPCheckScreen.....	648
DO_PSV_clean.....	725	Do_SAPCheckStatus.....	648
DO_PSV_close.....	726	Do_SAPCheckTitle.....	649
DO_PSV_commit.....	727	Do_SAPClearMessages.....	649
DO_PSV_creocc.....	729	Do_SAPDumpEvent.....	650
DO_PSV_delete.....	730	Do_SAPExtractString.....	650
DO_PSV_fclose.....	731	Do_SAPFrontEnd.....	650
DO_PSV_fetch.....	732	Do_SAPFullMenu.....	651
DO_PSV_ffield.....	733	Do_SAPGetControlValue.....	651
DO_PSV_fopen.....	733	Do_SAPGetIt_Event.....	652
DO_PSV_fread.....	734	Do_SAPGetScreenTitle.....	652
DO_PSV_free.....	735	Do_SAPGetStatusMsg.....	652
DO_PSV_fwrite.....	736	Do_SAPInit.....	653
DO_PSV_init.....	737	Do_SAPLogging.....	653
DO_PSV_logoff.....	738	Do_SAPLogin.....	653

Do_SAPLogoff	654	DO_SQLBuildDataLink	429
Do_SAPSendDbClick	654	DO_SQLCancel	445
Do_SAPSendEvent	654	DO_SQLCloseCursor	445
Do_SAPSendMenu	655	DO_SQLColAttribute	446
Do_SAPSendOKCode	655	DO_SQLColumns	446
Do_SAPSendPFKey	655	DO_SQLConnect	448
Do_SAPSendReturn	656	DO_SQLCopyDesc	448
Do_SAPSetCheckScreenWildcard	656	DO_SQLDescribeCol	449
Do_SAPSetCtrlValue	656	DO_SQLDisconnect	450
Do_SAPSetCursor	657	DO_SQLDriverConnect	450
DO_SaveReplyType	835	DO_SQLEndTran	450
DO_ScalarBindA	489	DO_SQLExecDirect	451
DO_SetAssumedContentType	836	DO_SQLExecute	451
DO_SetBaudRate	836	DO_SQLFetch	452
DO_SetBaudRateEx	837	DO_SQLFreeConnect	452
DO_SetCheckpointName	837	DO_SQLFreeHandle	453
DO_SetCookie	837	DO_SQLFreeStmt	453
DO_SetCookieEx	838	DO_SQLGetConnectAttr	430
DO_SetJavascriptCleanupThreshold	839	DO_SQLGetCursorName	454
DO_SetMaxBrowserThreads	840	DO_SQLGetData	454
DO_SetMaximumRetries	840	DO_SQLGetDataLinkAttr	430
DO_SetRefreshTimeout	840	DO_SQLGetDescField	455
DO_SetRetryWait	841	DO_SQLGetDescRec	456
DO_SetSSLConnectString	672	DO_SQLGetEnvAttr	456
DO_SetTimeout	841	DO_SQLGetLength	431
DO_SetTransactionCleanup	628	DO_SQLGetPosition	432
DO_SetTransactionStart	629	DO_SQLGetStmtAttr	432
DO_SetValue	630	DO_SQLGetSubString	433
DO_SLEEP	631	DO_SQLGetTypeInfo	457
DO_SoftClose	490	DO_SQLNumResultCols	458
DO_SQLAllocConnect	442	DO_SQLParamData	434
DO_SQLAllocHandle	443	DO_SQLPrepare	459
DO_SQLAllocStmt	443	DO_SQLPutData	458
DO_SQLBindCol	444	DO_SQLRetrieveParamValue	459
DO_SQLBindFileToCol	425	DO_SQLRowCount	460
DO_SQLBindFileToParam	426	DO_SQLSetConnectAttr	460
DO_SQLBindParameter	436	DO_SQLSetConnection	434

DO_SQLSetConnectOption	461	Do_Tuxsetwsnaddr	687
DO_SQLSetCursorName.....	462	Do_Tuxstring.....	687
DO_SQLSetDescField.....	463	Do_TuxStrlenEncodeString.....	688
DO_SQLSetDescRec.....	463	Do_Tuxsubstr.....	688
DO_SQLSetEnvAttr	464	Do_Tuxtpabort.....	689
DO_SQLSetPos	465	Do_Tuxtpalloc	689
DO_SQLSetScrollOptions.....	465	Do_Tuxtpbegin	690
DO_SQLSetStmtAttr	466	Do_Tuxtpbroadcast.....	690
DO_SQLSetStmtOption.....	467	Do_Tuxtpcall	691
DO_SQLSpecialColumns.....	468	Do_Tuxtpcommit	691
DO_SQLStatistics.....	469	Do_Tuxtpconnect	691
DO_SQLTables	469	Do_Tuxtpdiscon	692
DO_SQLTransact	470	Do_Tuxtpenqueue	693
DO_SSLReuseSession	673	Do_Tuxtpinit	694
DO_SSLUseCipher	674	Do_Tuxtppost	694
DO_SSLUseClientCert	675	Do_Tuxtprealloc	695
DO_SSLUseClientCertPass.....	675	Do_Tuxtprecv	695
DO_SSLUseProxy.....	676	Do_Tuxtpscmt	696
DO_StartAWL.....	645	Do_Tuxtpsend.....	696
DO_strdup.....	494	Do_Tuxtpsprio	696
DO_substr.....	470	Do_Tuxtpterm	697
Do_TuxAppendBuffer	679	Do_TuxUseCertificates	697
Do_TuxBufMemset	679	Do_Tuxxoctet	697
Do_TuxBuildBuffer	680	DO_URB_AsciiToHex	703
Do_Tuxcarray	680	DO_URB_Init	704
Do_Tuxcertsubstr	680	DO_URB_setoprretry.....	704
Do_Tuxencode.....	681	DO_URB_ubin2uf	704
Do_TuxFinit	682	DO_URB_udbl2uf.....	705
Do_TuxFMLData.....	682	DO_URB_ucreate.....	705
Do_TuxgetFMLData	683	DO_URB_uedelete.....	706
Do_TuxGetPartialBuffer	683	DO_URB_uentcreo	706
Do_Tuxgetrevent.....	684	DO_URB_uentoccs.....	707
Do_TuxgetTuxBuffer	684	DO_URB_uentseto	707
DO_TuxInitData.....	685	DO_URB_ufreeh	708
Do_TuxOutputBuffer	685	DO_URB_uinstdel	708
Do_TuxSet ViewData	686	DO_URB_uinstnew	708
Do_TuxSet ViewEnv.....	686	DO_URB_uinstopr	709

DO_URB_ulist2uf	709	DO_WSK_Getsockname.....	772
DO_URB_ulistdel.....	710	DO_WSK_HexDecode.....	772
DO_URB_ulistfree	711	DO_WSK_Init.....	773
DO_URB_ulistget.....	711	DO_WSK_Ioctlsocket	773
DO_URB_ulistnew	712	DO_WSK_IsReadable	773
DO_URB_ulistput	712	DO_WSK_IsWritable	774
DO_URB_ulistputlist	713	DO_WSK_Listen	774
DO_URB_ulistputx	713	DO_WSK_Quiet	775
DO_URB_ulong2uf.....	714	DO_WSK_Read.....	775
DO_URB_unifree.....	714	DO_WSK_Recv	775
DO_URB_uniname.....	715	DO_WSK_Recvfrom	776
DO_URB_uopract	715	DO_WSK_Reorder	776
DO_URB_uoprprms.....	716	DO_WSK_Select	777
DO_URB_uprmdir	716	DO_WSK_Send.....	778
DO_URB_uprmgeth	717	DO_WSK_SendAll	778
DO_URB_uprmtype.....	717	DO_WSK_Sendto	779
DO_URB_ustr2uf	718	DO_WSK_SetsockOpt	779
DO_URB_uuf2bin.....	718	DO_WSK_Shutdown	780
DO_URB_uuf2dbl.....	719	DO_WSK_Socket	780
DO_URB_uuf2list	719	DO_WSK_Write	780
DO_URB_uuf2long.....	720	DoubleClick, Citrix command	406
DO_URB_uuf2str	720	DownloadMediaFromASX	844
DO_UseEntityList.....	842	DownloadMediaRP	845
DO_UseNumericReferenceList	842	DownloadMediaWMP	846
DO_UsePersistentConnections.....	842	DownloadWindowsMedia	846
DO_UseProxy	843	duplicated frameset page.....	146
DO_UseProxyAutomaticConfiguration	843	dynamic windows, Citrix	22, 23, 70, 71
DO_VerifyDocTitle.....	844	E	
DO_WSK_Accept	768	EasyScript for Oracle Forms Server	85
DO_WSK_Bind	768	EcoTOOLS6	250
DO_WSK_Closesocket	769	Edit menu	200
DO_WSK_Connect	769	EditorDialog.....	542
DO_WSK_Expect	769	Emailing test data from QALoad Analyze	268
DO_WSK_ExpectAny	770	EnableStatisticsRP.....	846
DO_WSK_ExpectAnyExpr	770	encoding DBCS.....	168
DO_WSK_ExpectExpr.....	771	End_Transaction	631
DO_WSK_GetSocket.....	771	END_UENTITY.....	721

EndBlock	423	GET_HOME_DIR.....	634
EndCheckpoint	632	GET_LOGFILES_DIR	634
ENTITY_LIST	797	GET_RELATIVE_VUNUM	635
error handling.....	193, 203	GET_SCRIPTS_DIR.....	635
EscapeStr	781	GET_TIMINGFILES_DIR.....	635
Execute.....	287, 298	GetAbsolutePage.....	357
EXIT.....	632	GetAbsolutePosition	357
exporting data to html.....	267	GetActiveCommand	358
exporting rip files.....	268	GetActiveConnection	344, 358
exporting test data	267	GetActualSize.....	310
extract string, Visual Navigator script item	145	GetAttributes.....	298, 310, 327, 337
ExtractVariantValue.....	400	GetBindColumnData.....	471
F		GetBOF.....	359
File menu	200, 246	GetBookmark	359
files		GetCacheSize	359
Visual Scripting.....	155	GetCharset	390
Fill In Form		GetChildren	344
Page sub-item	143	GetChunk	311
Fill In Form	150	GetCollect	360
Fill_In	847	GetCommandStream	287
Find and Replace dialog box, visual scripting .	166	GetCommandText	288
Find, ADO command	356	GetCommandTimeout	288, 299
floating toolbar	15	GetCommandType	289
Flush	389	GetConnectionString.....	299
FormCanvas.....	542	GetConnectionTimeout	300
forms, in Visual Navigator	150	GetControlValue.....	544
FormStatusBar	543	GetCount	322, 336, 340
FormWindow	543	GetCursorLocation	300, 360
Frames		GetCursorType.....	361
in Visual Navigator scripts.....	146	GetDataFormat	312
frameset	146	GetDataMember	361
G		GetDataSource	362
Get	849	GetDefaultDatabase.....	300
GET_ABSOLUTE_VUNUM	632	GetDefinedSize	312
Get_Data	633	GetDialect	289
GET_DATA_FIELD.....	633	GetDirection	328
GET_DATAPOOLS_DIR.....	633	GetEditMode.....	362

GetEOF.....	363	GetState.....	302, 347, 369, 392
GetEOS.....	390	GetStatus.....	315, 369
GetFields.....	345, 363	GetStayInSync.....	370
GetFilter	364	GetString.....	370
GetIndex	364	GetType.....	315, 338, 393
GetIsolationLevel	301	GetUnderlyingValue.....	316
GetItem	323, 336, 341	GetValue	316, 330, 339
GetLineSeparator.....	391	GetVersion	303
GetLocalAddr	781	glossary	865
GetLocalPort	782	graph	
GetLockType.....	364	bar chart.....	255
GetMarshalOptions.....	365	cumulative response time distribution	255
GetMaxRecords.....	365	customizing.....	257
GetMode	301, 345, 391	line.....	255
GetName	290, 312, 328, 338	response time distribution.....	255
GetNamedParameters	290	Graph Toolbar buttons.....	249
GetNewEnum	320, 337, 341	graphing	
GetNumericScale.....	313, 329	bar	255
GetOriginalValue	313	checkpoints.....	255
GetPageCount	366	counters.....	256
GetPageSize	366	cumulative response time distribution	255
GetParameters.....	290	datapoints.....	255
GetParentURL	346	line.....	255
GetPosition	391	Player Performance Counters.....	256
GetPrecision	314, 329	response time distribution.....	255
GetPrepared.....	291	Server Monitoring Data.....	256
GetProperties.....	291, 314, 366	Top Processes.....	257
GetProvider	302	transaction throughput	255
GetRecordCount	367	graphing.....	255
GetRecordType.....	346	graphing.....	255
GetRemoteAddr.....	783	group	
GetRemotePort	783	checkpoints.....	250
GetRows	367	ClientVantage.....	250
GetSize.....	330, 392	counters.....	250
Getsockname.....	783	EcoTOOLS6.....	250
GetSort	368	player performance counters.....	250
GetSource	347, 368	remote monitoring.....	250

reports.....	250	WinSock .	767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790
RIP Files.....	250	WWW	798, 799, 801, 803, 804, 805, 807, 808, 809, 811, 812, 814, 815, 816, 817, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 849, 851, 852, 854, 859, 860, 861, 862
server analysis.....	250	ADO	285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400
server monitoring	250	group, definition.....	250
ServerVantage	250	H	
Top Processes	250	heartbeat message.....	238
group	250	Help menu	200
Group		HelpDialog.....	545
ADO		HiByte	784
DB2 .	424, 425, 426, 427, 429, 430, 431, 432, 433, 434	HTM	245
ODBC	435, 436	HTML character entities.....	797
ODBC/DB2.....	442, 443, 444, 445, 446, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470	HTML numeric references	797
Oracle 7...	475, 476, 477, 478, 479, 480, 481, 482, 485, 486, 487, 488, 489, 490	HTML Page form	143
Oracle 7/8	492, 493, 494	HTML Page items.....	143
Oracle 8...	501, 502, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524	HTML reports.....	267
Oracle Forms Server	534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 610, 611, 612, 613, 614, 615, 616, 617, 618	HTTP headers	
QALoad ...	622, 623, 625, 626, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 642, 643, 644, 645	in Visual Navigator scripts.....	149
QARun Integration	645	inserting.....	157
SAP..	648, 649, 650, 651, 652, 653, 654, 655, 656, 657	HTTPConnectToFormsServlet	545
SSL.....	672, 673, 674, 675, 676	HTTPConnectToListenerServlet.....	546
Tuxedo	679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697	HTTPInitialFormsConnect	546
UNIFACE.	703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720	HTTPReceiveMessage.....	547
UNIFACE Polyserver	724, 725, 726, 727, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 745, 746, 749, 750, 751, 752, 754, 755, 756, 757, 758, 760, 761, 762	HTTPS	180

HTTPSetHdrProperty	548	licensing virtual users.....	213
HTTPSetListenerServletParms	548	ListValuesDialog	552
HTTPSetURL.....	549	load test	
HTTPXmitMsg.....	549	adding Players.....	208
HTTPXmitTerminalMessage	549	running.....	197
I		load test.....	208
ICA file	68, 69	Load Test Summary	
IconButton	550	sample.....	258
ImageItem	551	Load Test Summary	258
increment variable, Visual Navigator script item	142	load-balanced environment, Citrix	
inserting commands.....	17	scripting techniques.....	24, 73
inserting commands in a capture file.....	16	load-balanced environment, Citrix	66
installing the NetLoad server module.....	183	LoadFromFile	393
installing UNIX Players.....	240, 242	LoByte	784
integration		Log	785
Application Expert	229	LOG_ERROR.....	636
Application Vantage.....	229	LogComment	552
ServerVantage	231	logfile generation	211
integration/Application Expert.....	228	logfiles.....	64
integration/Application Vantage.....	228	Logging	553
interface	191	LogonDialog	553
IP addresses		M	
Application Expert	229	Machine Assignment tab	197
Application Vantage.....	229	Machine Assignments, in Session report	259
IP Spoofing		Machine Configuration	
Visual Navigator script item	141	saving.....	207
J		Machine Configuration tab	194
Java.....	79	Machines in Use, in Session report.....	259
Java applet.....	58, 175	managing datapoints.....	255
JavaContainer	551	menu	
JavaDoc	79	Analyze	246
JavaScript.....	56, 79, 173	Help	249
K		Oracle Variablization	82
KeyDown	407	View	247
KeyUp.....	407	Visual Scripting.....	135
L		menu.....	14
launching Analyze	250	menu.....	239

menu	246
MenuInfo	554
MenuParametersDialog.....	554
merge ServerVantage file.....	248
monitoring CPU usage.....	201
Monitoring Options tab.....	196
MouseDown	408
MouseMove.....	408
MouseUp	409
Move	371
MoveFirst.....	371
MoveLast	372
MoveNext.....	372
MovePrevious.....	373
MoveRecord	347
MyByteOrder	785
N	
native character support	168
Navigate_To	851
NavigateTo	143
NetLoad	
creating a datapool	184
editing a datapool	185
entering/editing a datapool description	185
installing the server module.....	183
starting a session	184
starting the server module.....	183
NetLoad.....	183
new variable.....	162
NextField.....	325
NextParameter.....	326
NextProperty	324
NextRecordset	373
node-locked license.....	213
NTLM authentication, in Visual Navigator scripts	149
NUMERIC_REFERENCE_LIST.....	797

O	
OctalToChar	636
ODBC	435
ODBC/DB2.....	438
ofsActivateListItem	555
ofsActivateTreeItem	556
ofsActivateWindow	556
ofsClickButton	557
ofsClickTextFieldItem	558
ofsClosePopList	559
ofsCloseWindow	559
ofsCollapseTreeItem	560
ofsColorAdd	561
ofsConnectToSocket	561
ofsDeActivateWindow	562
ofsDefineTreeNode	562
ofsDefineTreeNodeOffset.....	563
ofsDelconifyWindow	564
ofsDeSelectItem	565
ofsDeselectTreeEvent	565
ofsEdit	566
ofsExpandTreeItem	567
ofsFindLOVValue.....	567
ofsFocus	568
ofsHideWindow	569
ofsHTTPConnectToFormsServlet	569
ofsHTTPConnectToListenerServlet	570
ofsHTTPDisconnect	570
ofsHTTPInitialFormsConnect	571
ofsHTTPDoSSLHandshake.....	570
ofsHTTPSetHdrProperty	570
ofsHTTPSetListenerServletParms	571
ofsIconifyWindow	571
ofsIndexKey	572
ofsIndexSKey.....	573
ofsInitSessionCmdLine.....	574
ofsInitSessionTimeZone.....	574

ofsListItemValue	575	ofsSetPropertyFloat	597
ofsLoadValue.....	575	ofsSetPropertyInteger.....	598
ofsLOVRequestRow	576	ofsSetPropertyPoint	598
ofsLOVSelection	577	ofsSetPropertyRectangle	599
ofsMenuParamDlgOK	577	ofsSetPropertyString	600
ofsOpenWindow	578	ofsSetPropertyStringArray.....	600
ofsRemoveFocus.....	579	ofsSetPropertyVoid	601
ofsScroll	579	ofsSetRequiredVAList	602
ofsScrollSize.....	580	ofsSetRunOptions.....	602
ofsSelectedItem	580	ofsSetScaleInfo	603
ofsSelectMenuItem	581	ofsSetScreenResolution	604
ofsSelectTreeEvent	582	ofsSetSelection	604
ofsSendRecv	582	ofsSetServerFailedMsg.....	605
ofsServerSideDisconnect	583	ofsSetServletMode.....	605
ofsSetColorDepth	583	ofsSetValue.....	606
ofsSetCursorPosition	584	ofsSetWindowLocation	606
ofsSetDisplaySize.....	584	ofsSetWindowSize.....	607
ofsSetErrorDialogTitle.....	585	ofsShowWindow	608
ofsSetExpectedServerMsg.....	585	ofsSocketDisconnect	608
ofsSetFontName.....	586	ofsStartSubMessage.....	609
ofsSetFontSize	587	ofsTabControlTopPage	609
ofsSetFontStyle.....	587	ofsUnSetPropertyBoolean	610
ofsSetFontWeight	588	open	303, 348, 374, 394
ofsSetICXTicket	588	OPEN_DATA_POOL.....	637
ofsSetInitialVersion	589	opening a timing file.....	250
ofsSetJavaContainerArgName.....	590	OpenSchema.....	304
ofsSetJavaContainerArgValue.....	590	options	
ofsSetJavaContainerEvent	591	Workbench	
ofsSetLogonDatabase	591	directory	12
ofsSetLogonPassword	592	file.....	12
ofsSetLogonUserName	593	options.....	12
ofsSetNoRequiredVAList	593	Oracle	
ofsSetPropertyBoolean	594	conversion options.....	80
ofsSetPropertyByte	595	optimizing Player for	240
ofsSetPropertyByteArray	595	recording options.....	80
ofsSetPropertyCharacter.....	596	Oracle.....	80
ofsSetPropertyDate.....	596	Oracle.....	80

Oracle.....	240	PageCheck.....	143
Oracle 7.....	471	parameters.....	242, 243
Oracle 7/8.....	491	password-protected directory.....	62, 179, 807
Oracle 8.....	495	Ping.....	409
Oracle command reference.....	85	Player	
Oracle Forms Server		default parameters.....	242
C++ scripts		errors.....	211
application statements.....	32, 95	log files.....	64
connection statements.....	31, 94	optimizing.....	240
debugging.....	34, 98	overview.....	239
disconnect statements.....	34, 98	startup.....	243
moving the transaction loop.....	35, 98	transfer UNIX scripts.....	186, 242
OFS and WWW Universal sessions....	37, 100	Player.....	240
C++ scripts.....	30	Player.....	242
C++ scripts.....	94	Player.....	243
conversion options.....	87	player performance counters	
overview.....	85	graphing.....	256
playback error codes.....	88, 233	player performance counters.....	253
recording options.....	86	Players	
server-side recording.....	85	adding to a test.....	208
Oracle Forms Server.....	79	Players.....	208
Oracle Forms Server.....	86	PlayMedia.....	851
Oracle Forms Server.....	87	Point.....	410
Oracle Forms Server.....	525	PopList.....	612
Oracle Forms Server method reference.....	94	PopupHelp.....	612
OracleAppsLogin.....	610	Post_To.....	852
OracleForms.....	611	PostTo.....	143
OracleFormsMsg.....	611	Pre-defined reports	
Output report.....	261	Client Throughput.....	258, 262
overview.....	245	Concurrent Users.....	258
overview of Application Expert integration....	228	Output.....	258
overview of Application Expert/Application		Player Performance.....	266
Vantage integration.....	228	Response Time Analysis.....	258
Overview of the QALoad Conductor.....	191	Server Monitoring.....	258
P		Summary.....	258
pacing		Top Ten Longest Checkpoint Durations.....	263
field on Script Assignment tab.....	193	Transaction Throughput.....	258
Page items.....	143		

PrintVariant	401	PutPrepared.....	295
PromptList.....	613	PutProvider	308
PutAbsolutePage.....	374	PutRefActiveConnection	296, 350, 381
PutAbsolutePosition	375	PutRefDataSource.....	382
PutActiveConnection	292, 349, 375	PutRefSource.....	382
PutAttributes.....	304, 316, 331, 339	PutSize.....	333
PutBookmark.....	376	PutSort	383
PutCacheSize.....	376	PutSource	350, 383
PutCharset.....	395	PutStayInSync.....	383
PutCollect.....	377	PutType.....	319, 334, 396
PutCommandText.....	292	PutValue.....	319, 334, 340
PutCommandTimeout	293, 305	Q	
PutCommandType.....	293	QALoad	619
PutConnectionString	305	QALoad Analyze Menus	246
PutConnectionTimeout	306	QALoad can't find TUXDIR environment variable	188
PutCursorLocation	306, 377	QALoad Conductor.....	250
PutCursorType	378	QALoad Player Main Window	240
PutDataFormat	317	QALoad Player Menus	239
PutDataMember	378	QALoad Script Development Workbench	
PutDefaultDatabase.....	306	menus.....	14
PutDefinedSize.....	317	toolbar buttons.....	14
PutDialect	294	QALoad Script Development Workbench	14
PutDirection	331	QALoad Script Development Workbench	250
PutFilter.....	379	QARun Integration	645
PutIndex.....	379	QARun scripts	
PutIsolationLevel	307	automatically generate.....	187
PutLineSeparator	395	manually generating.....	188
PutLockType.....	379	QARun scripts.....	187
PutMarshalOptions.....	380	R	
PutMaxRecords	380	RadioButton	613
PutMode.....	307, 350, 396	ramp-up session	215
PutName.....	294, 332	RandNumString.....	853
PutNamedParameters.....	295	RANDOM_NUMBER.....	637
PutNumericScale.....	318, 332	RANDOM_STRING	638
PutPageSize.....	381	Read	397
PutPosition	396	read datapool, Visual Navigator script item	141
PutPrecision	318, 333	READ_DATA_RECORD	638

ReadText.....	397	comma-separated value.....	267
recording		reporting.....	267
Java.....	79	reports	
load tests.....	219	Analyze	
multiple middleware sessions.....	13	Client Throughput.....	262
Oracle Forms Server.....	85	Concurrent Users.....	260
recording.....	13	Output.....	261
Recording middleware calls.....	15	Player Performance.....	266
recording options		Response Time Analysis.....	260
ADO.....	66	Server Monitoring.....	262
Citrix.....	68	Session.....	259
IIOP.....	120	Summary.....	258
Oracle.....	80	Top Ten Longest Checkpoint Duration ..	263
Oracle Forms Server.....	86	Transaction Throughput.....	262
SAP.....	101, 105	Analyze.....	258
TUXEDO.....	111	Application Expert.....	266
UNIFACE.....	116	Application Vantage.....	266
Winsock.....	120	CSV.....	268
WWW.....	170	integration.....	266
recording options.....	66	reports, how to view.....	269
recording options.....	68	reports, pre-defined .	258, 260, 261, 262, 263, 266
recording options.....	80	ReQuery.....	384
recording options.....	86	reset variable, Visual Navigator script item	142
recording options.....	101	Response.....	785
recording options.....	111	Response Time Analysis report.....	260
recording options.....	116	ResponseLength.....	786
recording options.....	120	RESTART_TRANSACTION_BOTTOM.....	854
recording options.....	170	RESTART_TRANSACTION_TOP.....	854
Recording toolbar.....	15	restarting transactions	
Refresh.....	323, 337, 342	DO_SetTransactionClean up.....	628
Region.....	853	DO_SetTransactionStart.....	629
Remote Monitoring		enabling	
counters.....	222	from the Script Assignment tab.....	193
removing datapool data.....	210, 216	SAP scripts.....	38, 107
renaming datapool variables.....	162	Resync.....	324, 384
Replay a script with PeopleSoft certificates	113	RMI.....	79
reporting		RND_DELAY.....	639

RND_DELAY_RANGE.....	639	SAPGuiCmd0	661
RollbackTrans.....	308	SAPGuiCmd1	662
RR_FailedMsg	639	SAPGuiCmd1Coll	662
RR_GetDebugFlag.....	640	SAPGuiCmd1Elmnt	663
RR_printf	640	SAPGuiCmd1Sub	664
Run menu	200	SAPGuiCmd1Sub1	664
Runform	614	SAPGuiCmd2	665
Running a Load Test	213	SAPGuiCmd3	666
running a simple test	243	SAPGuiConnect	666
running a test		SAPGuiCreateColl	667
Details view	198	SAPGuiDestroyColl	667
running a test	197	SAPGuiPropIdStr	668
running a test	231	SAPGuiPropIdStrExists.....	668
Running Scripts, in Session report	259	SAPGuiPropIdStrExistsEnd	669
runtime data transfer	215	SAPGuiSessionInfo.....	669
Runtime window		SAPGuiSetCheckScreenWildcard	670
Details view	198	SAPGuiVerCheckStr	671
Session view	199	Save	385
Runtime window	197	SaveToFile	398
Runtime Window menu	200	saving machine configurations	207
S		ScanExpr	786
sample Load Test Summary	258	ScanFloat	787
SAP		ScanInt	787
conversion options.....	102, 106	ScanLenString.....	788
handling multiple logons.....	41, 110	ScanRewind.....	788
post-test log files.....	103	ScanSkip	789
recording options.....	101, 105	ScanString	789
scripting techniques	41, 110	script	
troubleshooting	189, 190	adding messages for playback	17
SAP	101	inserting a datapool	157, 161
SAP	102	Java	79
SAP command index, Version 6.2	657	transfer.....	186, 242
SAP command index, Versions 4.x	646	script	186
SAPGui_error_handler.....	659	script	242
SAPGuiApplication.....	659	Script Assignment tab.....	193
SAPGuiCheckScreen	660	Script Assignments, in Session Report	259
SAPGuiCheckStatusBar	661	script conversion	154

script debugging	
ActiveData for Oracle.....	83
script debugging.....	211
Script Development Workbench	
configuring	12
starting.....	248
script editing	
inserting commands.....	17
inserting Visual Navigator items	156
techniques	
Citrix.....	22, 23, 24, 70, 71, 72, 73
general	18, 19
SAP.....	38, 41, 42, 107, 110, 111
Tuxedo	42, 43, 44, 113, 114, 115
Winsock	45, 47, 50, 51, 53, 121, 123, 126, 127, 129
script elements, Visual Navigator	137
script validation	243
SCRIPT_MESSAGE.....	643
server analysis.....	253, 256
Server Analysis Agents	
setting up monitoring.....	227
Server Analysis Agents	201, 227
server farm, Citrix	
scripting techniques	24, 73
server farm, Citrix	66
Server Monitoring	
Remote Monitoring	221
ServerVantage	230
Top Processes.....	257
Server Monitoring	196, 201
Server Monitoring	253
Server Monitoring Data	
graphing.....	256
Server Monitoring Data	253
Server Monitoring Report	262
server replies	
Winsock.....	51, 53, 127, 129
ServerVantage	201, 231, 250, 253, 256, 262
service level threshold	
option on Script Assignment tab.....	193
Session report.....	259
Set	854
SET_ABORT_FUNCTION	642
SetApplication.....	410
SetCitrixPort	411
SetConnectTimeout	411
SetDisconnectTimeout.....	411
SetDomainLoginInfo	412
SetEnableCounters.....	412
SetEnableWildcardMatching.....	413
SetEOS.....	398
SetExpectedServerMsg	614
SetHeartbeat	615
SetICAFile.....	413
SetLoginInfo	413
SetPingTimeout.....	414
SetProxy	615
SetTimeout	789
Setting a default middleware session	12
setting Conductor options.....	203
Setting Up a Test	192, 202
Setting up a test session ID	203
Setting up QALoad to run Oracle scripts on UNIX	84
Setting up QALoad to run Tuxedo scripts on UNIX	112
SetTyperate.....	790
SetWaitPointTimeout	414
SetWindowMatchName.....	414
SetWindowMatchTitle.....	414
SetWindowRetries.....	415
SetWindowTimeout	415
SetWindowVerification	415
ShowMediaRP	859

simulate	
browser caching	62, 179
CGI requests.....	56, 173
cookie.....	60, 177
frame.....	59, 176
Java applet.....	58, 175
JavaScript	56, 173
password-protected directories.....	62, 179
static HTML page.....	62, 179
Visual Basic script	58, 174
SkipExpr	790
SkipLine.....	399
SLEEP	
as a component of checkpoint duration	245
Visual Navigator item	143
SLEEP	143
SLEEP	643
sleep factor	
field on Script Assignment tab	193
socket resources.....	189
Sort Grid dialog box	254
sorting data	254
SSL	180, 671
SSL scripts.....	181, 231
starting a test.....	231
Starting Conductor from the command line...	214
starting the NetLoad server module	183
startup parameters	242
statistics.....	221
streaming media	
streaming media support	167
Visual Navigator.....	132
streaming media.....	167
stripping datapools	210, 216
SubRequests	
Additional	143
SubRequests.....	143
Summary report	258
summary test results.....	199
Supports.....	386
SYNCH	644
synch, Visual Navigator script item.....	141
SYNCHRONIZE	644
T	
TabControl.....	616
technical support	ii
technical support	864
technical support	874
test	
starting.....	231
statistics	221
test.....	221
test.....	231
test information	259
Test Information Window	192
Test Options menu	200
test results	
checkpoint details.....	251
emailing.....	268
Load Test Summary	258
sorting.....	254
viewing	270
test results.....	251
test results.....	254
test results.....	258
test results.....	268
test results, thinning the data.....	203
TextArea.....	616
TextField	617
The Default Session Prompt didn't open?.....	188
thinning data points.....	255
thinning test data	
procedure.....	255
thinning test data	203

thresholds.....	217, 218	troubleshooting	190
TIM	245	Tuxedo	
timing data, thinning out.....	203	conversion options.....	112
timing file		recording options	111
troubleshooting	232	Tuxedo	111
timing updates.....	215	Tuxedo	112
tips		Tuxedo	676
UNIX	231	Tuxedo command reference.....	113
tips.....	231	Type	416
Tlist.....	617	TypeChar	416
toolbar		TypeVK.....	417
graph	249	U	
Recording.....	15	UFIELD	721
toolbar	200	UnEscapeStr	790
toolbar	249	UNIFACE	
toolbar buttons	14, 249	conversion options.....	116
Tools menu	248	recording options	116
Top Processes		UNIFACE Polyserver	722
detail data	254	Universal session	
graphing.....	257	WWW/Oracle Forms Server.....	37, 100
traffic filters		Universal session.....	13
in Visual Navigator	139	UNIX	
transaction cleanup		installing Players.....	240, 242
Visual Navigator item	141	running tests.....	231
transaction duration		transfer scripts	186, 242
understanding durations.....	245	UNIX	186
transaction loop	131	UNIX	242
transaction throughput		UNIX/DB2 playback	186, 243
report	262	Update.....	324, 386
Tree.....	618	UpdateBatch	387
troubleshooting		Using Integrated Server Monitoring with QALoad	227
Performance issues with SAP or Citrix scripts	190	Using the debug window	216
QALoad can't find TUXDIR environment variable	188	using the Function Wizard	17
SAP script validation fails	189	V	
The default session prompt didn't open.....	188	validation	
Winsock running out of socket resources ...	189	procedure	
		Conductor	211

Player	243	forms.....	150
VARDATA.....	645	frames	146
variables		HTML Page form	143
adding.....	162	HTML Page items.....	143
naming.....	162	inserting script items.....	156
replacing in Visual Navigator	166	interface.....	134
Visual Navigator.....	160	menus.....	135
variablization		overview.....	130
Visual Navigator.....	131	recording a script	155
Winsock scripts.....	47, 123	replacing variables.....	166
Variablization menu	82	script elements.....	137
Variablize dialog box	82	streaming media support.....	132
Verify.....	860	transaction loop.....	131
verifying checkpoints.....	65	tree-view	137
view	245	variables.....	160
View Menu	200, 247	variablization	131
viewing		Web Playback Options form.....	138
reports.....	269	XML	152, 164
test results.....	270	XML support.....	132, 162, 167
viewing.....	270	visual scripting.....	130
Viewing datapool usage.....	217	W	
viewing log files.....	211	WaitForCaptionChange.....	417
viewing rip files.....	211	WaitForScreenUpdate	417
virtual user licensing.....	213	WaitForWindowActivate	418
Virtual User menu	200	WaitForWindowCreate.....	418
virtual users		WaitForWindowDestroy	419
adding on-the-fly	202	WaitForWindowMinimize.....	419
Visual Basic script.....	58, 174	WaitForWindowMove	420
Visual Navigator		WaitForWindowResize.....	420
Action item	149	WaitForWindowStyleChange.....	421
client certificate	140	Web browser	
converting a script	154	configure.....	167
datapools.....	160	viewing test results.....	270
DBCS.....	133	Web Playback Options form	138
files.....	155	Winsock	
Find and Replace.....	166	character representation	45, 121
form elements.....	150	command reference.....	121

recording options.....	120	WWW	791
script	117	WWW command reference.....	171
server replies	51, 53, 127, 129	WWW scripts.....	137
variablization	47, 123	WWW session	
Winsock	120	Visual Scripting.....	134
Winsock	762	WWW_FATAL_ERROR.....	861
Winsock running out of socket resources.....	189	X	
Workbench		X_Coord.....	861
starting.....	248	XmitMsg	618
Workbench	248	XmitTerminalMessage.....	618
Workspace.....	245	XML	
Write	399	report format, Analyze.....	245
WriteText	400	XML requests.....	163
WWW		XML support.....	132, 162, 167
conversion options.....	169	XmlRequest.....	862
inserting script items manually	156	XSL.....	245
recording options.....	170	Y	
streaming media	167	Y_Coord	862
XML support	132, 162, 167	Z	
WWW	169	zip file, creating in Analyze.....	268
WWW	170		