Language Reference Commands

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

Technical Support Hotline:
1-800-538-7822

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

Table Of Contents

QALoad Online Help

Language Reference Commands

List of QALoad Language Reference Commands

The QALoad Language Reference provides command reference information for general and middleware-specific commands. Select one of the middleware groups to display from the following list:

.NET

Citrix

ODBC

Oracle Forms Server

QALoad

SAP

SSL

TestPartner

Winsock

WWW

.NET

.NET Commands

BeginCheckpoint

Marks the beginning of a checkpoint.

Checkpoint Pair
Inserts a checkpoints around the text you select.

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.

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.

DO_MSLEEP
Inserts a sleep for the number of seconds defined in the parameter.

DO_SLEEP
Inserts a sleep for the number of seconds defined in the parameter.

EndCheckpoint
Indicates the end of a checkpoint, corresponding to a BeginCheckpoint command.

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_RELATIVE_VUNUM
Gets the relative virtual user number.

GET_TOTAL_TRANSACTIONS
Indicates the total number of transactions that will be performed. The value should be the same as the Transaction Count specified for the script from within the Conductor.

GET_TRANSACTION_NUMBER
Indicates the current transaction number that the Virtual User is performing. The value should be between zero and the Transaction Count specified for the script from within the Conductor.

OPEN_DATA_POOL
Opens the datapool file.

OPEN_SHARED_DATA_POOL
Opens the shared datapool file.

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__printf
Prints formatted output to the standard output stream.

SCRIPT_MESSAGE
Inserts custom script messages into a timing file during test execution.

SLEEP
Pauses a script for the specified number of seconds. This command is not affected by the sleep factor percentage specified in QALoad Conductor.

VARDATA
Replaces a string with a datapool variable.

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 ( char* CheckpointName );
```

Return Value

Parameters

Parameter	Description
CheckpointName	String containing a description of the checkpoint. This value cannot be longer than 127 characters.

Example

```
BeginCheckpoint("Testing User-defined");
DO_Http("GET http://compuweb.compuware.com/ HTTP/ 1.0\r\n\r\n");
EndCheckpoint("Testing User-defined");
```

Checkpoint Pair

Inserts a Begin Checkpoint item before the currently selected HTML Page and an End Checkpoint after the currently selected HTML Page.

Checkpoints are used to measure duration times for certain actions to be completed. You can move either the Begin or End checkpoint item to encompass several requests, if necessary. To move either item, highlight it and then click Move Up/Move Down in the form-view.

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 ();
```

Return Value

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);
```

Return Value

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.

Versions

Versions of COUNTER_VALUE are:

COUNTER_VALUE (int Counter_ID, long Counter_Value);

COUNTER_VALUE (int Counter_ID, float Counter_Value);

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, CounterDataTypeEnum
Data_Type, CounterCounterTypeEnum Counter_Type );
```

Return Value

any value except -1 if successful
-1 if unsuccessful

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	*CounterDataTypeEnum* Counter data type. Valid values are: Value — Description DATA_LONG — The counter values will be of type long. DATA_FLOAT — The counter values will be of type float.
Counter_Type	*CounterCounterTypeEnum* Counter type. Valid values are: Value — Description COUNTER_CUMULATIVE — The counter data is cumulative. COUNTER_INSTANCE — The counter data is 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 );
}
```

DO_MSLEEP

Inserts a sleep for the number of seconds defined in the parameter.

The parameter passed to DO_MSLEEP 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_MSLEEP 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_MSLEEP command is encountered, it sleeps for a random time frame ranging from 0 to the value specified.

Syntax

```
DO_MSLEEP( int nMilliseconds );
```

Return Value

Parameters

Parameter	Description
nMilliseconds	Number of milliseconds 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_MSLEEP( 5 );  /* Sleep 5 seconds */
```

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( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	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 */
```

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 ( char* CheckpointName ) ;
```

Return Value

Parameters

Parameter	Description
CheckpointName	String containing a description of the checkpoint. This value cannot be longer than 127 characters.

Example

```
BeginCheckpoint("Testing User-defined");
DO_Http("GET http://compuweb.compuware.com/ HTTP/ 1.0\r\n\r\n");
EndCheckpoint("Testing User-defined");
```

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 ();
```

Return Value

int -- absolute virtual user number

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 ();
```

Return Value

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 one (1).

Syntax

```
GET_DATA_FIELD (int datapool ID, int FieldNum);
```

Return Value

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 one (1).

Example

```
BeginCheckpoint();
RR__printf("Datapool Entry #1: %s", GET_DATA_FIELD (1, 1) );
DO_SLEEP(500);
EndCheckpoint(1);
```

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 ();
```

Return Value

int -- relative virtual user number

Parameters

None.

Example

```
int vunum;
nuvum = GET_RELATIVE_VUNUM();
RR__printf("I am vu %d", vunum);
```

GET_TOTAL_TRANSACTIONS

Indicates the total number of transactions that will be performed. The value should be the same as the Transaction Count specified for the script from within the Conductor.

Syntax

int GET_TOTAL_TRANSACTIONS()

Return Value

The total number of transactions as an Integer.

Parameters

None

Example

```
int x;
x = GET_TOTAL_TRANSACTIONS();
```

GET_TRANSACTION_NUMBER

Indicates the current transaction number that the Virtual User is performing. The value should be between zero and the Transaction Count specified for the script from within the Conductor.

Syntax

int GET_TRANSACTION_NUMBER()

Return Value

The current transaction number as an Integer.

Parameters

None

Example

```
int x;
x = GET_TRANSACTION_NUMBER();
```

OPEN_DATA_POOL

Opens the datapool file.

This command line is typically placed before the BEGIN_TRANSACTION( ) statement.

Syntax

```
OPEN_DATA_POOL (char* filename, int poolNumber, unsigned short fileAction);
```

Return Value

TRUE if the file was successfully opened; FALSE otherwise.

Parameters

Parameter	Description
filename	The name of the datapool file, including a drive and path.
poolNumber	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.
fileAction	Action to be taken when the "End of File" is reached. DPA_REWIND_AT_EOF If the datapool file should be rewound to the beginning after it reaches the end. DPA_DONT_REWIND if the datapool file should not be rewound.

Example

```
OPEN_DATA_POOL( "C:\\Program Files\\Compuware\\ QALoad \\Middlewares\\
   SQLServer\\Scripts\\junk.dat", SS_1, DPA_REWIND_AT_EOF );
```

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( int datapool_ID );
```

Return Value

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 ( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	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__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]...);
```

Return Value

The number of characters printed or a negative value if an error occurs.

Parameters

Parameter	Description
format	Format control.
argument	Optional arguments.

Example

```
/* This code segment shows examples of the usage of the RR__printf function to produce
formatted output for various datatypes.  */

char ch='h', *string="computer";
int count=-9234;
double fp=251.7366;
wchar_t wch=L'w', *wstring=L"Unicode";

/*Display integers. */
RR__printf("Integer formats:\n" "\tDecimal: %d Justified: %.6d Unsigned: %u\n", count,
count, count, count);

RR__printf("Decimal %d as:\n\tHex: %Xh C hex: 0x%x Octal: %o\n", count, count, count,
count);

/* Display in different radixes. */
RR__printf("Digits 10 equal:\n\tHex: %i Octal: %i Decimal: %i\n",0x10, 010, 10);

/* Display characters. */
RR__printf("Characters in field:\n%10c%5hc%5C%5lc\n", ch, ch, wch, wch);

/* Display strings. */
RR__printf("Strings in field:\n%25s\n%25. 4hs\n\t%S%25.3ls\n", string, string, wstring,
wstring);

/* Display real numbers. */
RR__printf("Real numbers:\n\t%f%.2f%e%E\n", fp, fp, fp, fp);

/* Display pointer. */
RR__printf("\nAddress as:\t%p\n", &count);

/* Count characters printed. */
RR__printf("\nDisplay to here:\n");
RR__printf("1234567890123456%n78901234567890\n", &count);
RR__printf("\tNumber displayed: %d\n\n", count);
```

Output

```
Integer formats:

                    Decimal: -9234
                    Justified: -009234
                    Unsigned: 4294958062

Decimal -9234 as:

                    Hex: FFFFDBEEh
                    C hex: 0xffffdbee
                    Octal: 37777755756

Digits 10 equal:

                    Hex: 16
                    Octal: 8
                    Decimal: 10

Characters in field:

                    h h w w

Strings in field:

                    computer
                    4hs
                    Uni

Real numbers:
```

```
                    251.736600
                    251.74 2.517366e+002

Address as:

                    0141FDC0
Display to here:

                    12345678901234567890123456789O
Number displayed:

                    16
```

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 rhobot_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 ( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	The number of seconds to sleep before execution proceeds.

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)
```

Return Value

Parameters

Parameter	Description
n	The datapool field number. Field numbering starts at 1.

Example

```
Do_TuxFMLData ( 8302, 0, VARDATA(1));
```

Citrix

Citrix Commands

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.

CtxConnectICA

Connects to the Citrix server using an ICA file.

CtxConnectPubApp

Connects to a published application on a Citrix server or Citrix server farm.

CtxConnectServer

Connects to a Citrix server and possibly an application on the server.

CtxDisconnect
Disconnects from the Citrix server.

CtxDoubleClick
Double-clicks the mouse at the current location.

CtxFullBitmapExists

Checks to see if the current full screen bitmap matches the hash code passed into the BitmapHash argument.

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.

CtxPartialBitmapExists

Checks to see if the current partial screen bitmap matches the hash code passed into the BitmapHash argument.

CtxPoint
Moves the mouse to the specified location on the screen.

CtxScreenEventExists
Waits for the specified screen update to occur at the specified coordinates.

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.

CtxSetDomainLoginInfo
Connects to the Citrix server with the specified user name, password, and domain.

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.

CtxSetGracefulDisconnect

Specifies whether or not a logoff should be issued before issuing a disconnect.

CtxWaitForFullBitmap

Waits for a full screen bitmap to match the hash code passed into the BitmapHash argument.

CtxWaitForPartialBitmap

Waits for a partial screen bitmap to match the hash code passed into the BitmapHash argument.

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.

CtxWaitForWindowLgIconChange
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(PLAYERINFO *pInfo, char *msg);
```

Return Value

Parameters

Parameter	Description
`pInfo`	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, CtxMouseButtonEnum button, CtxKeyModifierEnum mod);

Return Value

Parameters

Parameter	Description
`windowInfo`	Pointer to a Citrix Window Information object containing window data.
`holdTime`	Number of milliseconds to hold down the button.
`button`	*CtxMouseButtonEnum* A mouse button to use for this action Value — Description `NONE` — No keyboard modifier was specified `L_BUTTON` — The left mouse button `R_BUTTON` — The right mouse button `M_BUTTON` — The middle mouse button
`ModifierKeys`	*CtxKeyModifierEnum* The following modes are available: Value — Description `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);
```

CtxConnectICA

Connects to the Citrix server using an ICA file.

Syntax

void CtxConnectICA (char* pszICAFile)

Return Value

Parameters

Parameter	Description
pszICAFile	ICA file name, excluding path.

Example

```
/* Declare Variables */
const char *CitrixICAFilename = "Calculator2.ica";
BEGIN_TRANSACTION();
DO_SetTransactionStart();
CtxConnectICA(CitrixICAFilename);
```

CtxConnectPubApp

Connects to a published application on a Citrix server or Citrix server farm.

Syntax

void CtxConnectPubApp(char *pszPubAppName, char *pszCitrixServer)

Return Value

Parameters

Parameter	Description
pszPubAppName	Name of the published application.
pszCitrixServer	Name of the Citrix server.

Example

```
/* Declare Variables */
const char *CitrixServer      = "dtw-labcitrix2";
const char *CitrixPubAppname  = "Calc";
BEGIN_TRANSACTION();
DO_SetTransactionStart();
CtxConnectPubApp(CitrixPubAppname, CitrixServer);
```

CtxConnectServer

Connects to a Citrix server and possibly an application on the server.

Syntax

void CtxConnectServer (char *pszCitrixServer, char *pszApplication, char *pszWorkingDir)

Return Value

Parameters

Parameter	Description
pszCitrixServer	Name of the Citrix Server.
pszApplication	Name of the startup application.
pszWorkingDir	Working directory of the startup application.

Example

```
/* Declare Variables */
const char *CitrixServer      = "qaccitrix";
const char *CitrixApplication = "calc.exe";
const char *CitrixAppWorkDir  = "c:\\";
BEGIN_TRANSACTION();
DO_SetTransactionStart();
CtxConnectServer(CitrixServer, CitrixApplication, CitrixAppWorkDir);
```

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);
```

Return Value

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);
```

CtxFullBitmapExists

Checks to see if the current full screen bitmap matches the hash code passed into the BitmapHash argument and, if not, waits for the specified time for the bitmap to match.

Versions

Versions of CtxFullBitmapExists are:

BOOL CtxFullBitmapExists(char *BitmapHash, char *BitmapTitle, long lTimeout = 0)

BOOL CtxFullBitmapExists(char *BitmapHash, char *BitmapTitle)

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);
```

Return Value

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);

Return Value

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, CtxMouseButtonEnum button, CtxKeyModifierEnum mod, long Xpos, long Ypos);

Return Value

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
button	*CtxMouseButtonEnum* A mouse button to use for this action Value — Description 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	*CtxKeyModifierEnum* The following modes are available Value — Description 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);

Return Value

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, CtxMouseButtonEnum button, CtxKeyModifierEnum mod, long Xpos, long Ypos);

Return Value

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
button	*CtxMouseButtonEnum* A mouse button to use for this action Value — Description NONE — No mouse button was specified L_BUTTON — The left mouse button R_BUTTON — The right mouse button M_BUTTON — The middle mouse button

<table>
<tr><td>ModifierKeys</td><td>CtxKeyModifierEnum
The following modes are available
<table><tr><td>Value</td><td>Description</td></tr><tr><td>NONE</td><td>No keyboard modifier was specified</td></tr><tr><td>SHIFT</td><td>Shift key</td></tr><tr><td>CONTROL</td><td>Control key</td></tr><tr><td>ALT</td><td>Alt key</td></tr><tr><td>EXTENDED</td><td>An extended key</td></tr></table></td></tr>
<tr><td>Xpos</td><td>Move the mouse to this X coordinate.</td></tr>
<tr><td>Ypos</td><td>Move the mouse to this Y coordinate.</td></tr>
</table>

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);
```

CtxPartialBitmapExists

Checks to see if the current partial screen bitmap matches the hash code passed into the BitmapHash argument and, if not, waits for the specified time for the bitmap to match.

Versions

Versions of CtxPartialBitmapExists are:

BOOL CtxPartialBitmapExists(char *BitmapHash, char *BitmapTitle, int x, int y, int width, int height, long lTimeout = 0)

BOOL CtxPartialBitmapExists(char *BitmapHash, char *BitmapTitle, int x, int y, int width, int height)

CtxPoint

Moves the mouse to the specified location on the screen.

Syntax

void CtxPoint(long X, long Y);

Return Value

Parameters

Parameter	Description
X	X coordinate.
Y	Y coordinate.

Example

```
CtxPoint(509, 422);
```

CtxScreenEventExists

Checks to see if the specified screen event has already occurred and, if not, waits the specified time for the event to occur.

Syntax

BOOL CtxScreenEventExists(CitrixScreenEventTypeEnum EventType, int nmWait, const char* EventInfo);

Return Value

Parameters

Parameter	Description
EventType	*CitrixScreenEventTypeEnum* Citrix screen event. Valid values are: Value — Description EVT_STR_CTXSCREENUPDATE — ScreenUpdate
nmWait	Amount of time in milliseconds to wait for the event.
EventInfo	Coordinates and size of target screen update in the format: x y width height.

Example

```
CtxClick(CWI_2, 188, L_BUTTON, NONE); //1087322563.276

     if(CtxScreenEventExists(EVT_STR_CTXSCREENUPDATE,3000,"0 224 39 10"))
     BeginBlock();
          RR__printf("Screen Update Found Test1");
     EndBlock();
```

CtxSetCitrixPort

Sets the port for the Citrix client to use to connect to the server.

Syntax

void CtxSetCitrixPort (int port);

Return Value

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);

Return Value

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);

Return Value

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

Specifies user name, password, and domain to use on connection to the Citrix server.

Syntax

void CtxSetDomainLoginInfo (const char *username, const char *password, const char *domain);

Return Value

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);

Return Value

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);

Return Value

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.

CtxSetGracefulDisconnect

Specifies whether or not a logoff should be issued before issuing a disconnect. If this is set to false, the Citrix server may need to be properly configured to handle the lingering sessions.

Syntax

```
void CtxSetGracefulDisconnect(BOOL enable)
```

Return Value

N/A

Parameters

Parameter	Description

enable	TRUE – a logoff will be issued before issuing a disconnect. BOOL – No logoff will be issued before issuing a disconnect

Example

The following is an example of using CtxSetGracefulDisconnect

```
CtxSetGracefulDisconnect( TRUE );
```

CtxSetOutputMode

Sets the output mode of the current script.

Syntax

void CtxSetOutputMode(int iOutputMode)

Return Value

Parameters

Parameter	Description
iOutputMode	Output: OUTPUT_MODE_NORMAL OUTPUT_MODE_WINDOWLESS OUTPUT_MODE_RENDERLESS

Example

```
/* Declare Variables */
const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;
/* Citrix replay settings */
CtxSetOutputMode(CitrixOutputMode);
```

CtxSetWaitPointTimeout

Sets the number of seconds to wait for a wait point. This applies to all waitpoint timeouts other than window create and destroy waitpoint timeouts, which are set using CtxSetWindowTimeout.

Syntax

void CtxSetWaitPointTimeout (int timeout);

Return Value

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 CtxWaitForWindowCreate method of the Citrix Window Information object.

This name is used for comparison if wildcards have been enabled by the CtxSetEnableWildcardMatching method call earlier in the script. You can use the CtxSetWindowMatchTitle function as many times as necessary in a script to change the window title match string after the window is created.

Syntax

void CtxSetWindowMatchTitle (CtxWI* windowInfo, char* strWindowMatchName);

Return Value

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.
```

Note: The CtxSetWindowMatchTitle function can be used as many time as necessary in a script. The CtxSetWindowTitle function should appear each time the window title changes during capture after the window is created. When the window title contains variable content, such as transaction number or date and time, you can change the CtxSetWindowTitle command to CtxSetWindowMatchTitle and use wildcards for the variable content. To use this approach, you must enable the wildcard matching function using CtxSetEnableWildcardMatching(TRUE).

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);

Return Value

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 created and destroyed in the CtxWaitForWindowCreate and the CtxWaitForWindowDestroy waitpoint commands, respectively.

Syntax

void CtxSetWindowTimeout (int timeout);

Return Value

Parameters

Parameter	Description
`timeout`	Number of seconds to wait for a window to be created.

Example

```
CtxSetWindowTimeout (30);
```

CtxSetWindowTitle

Sets the name of a previously-created window in the Citrix Window Information object.

Syntax

```
void CtxSetWindowTitle(const CtxWI* windowInfo, const char *strWindowTitle);
```

Return Value

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object that contains window data.
strWindowTitle	Character string containing the new window title.

Example

```
CtxWI *CWI_7 = new CtxWI(0x20122, "Untitled - Notepad", 72, 54, 481, 322);
CtxWaitForWindowCreate(CWI_7, 500);
// Update the window title after Notepad loads the file;
// this is necessary so following script commands which reference window CWI_7
// can properly match the window title.
CtxSetWindowTitle(CWI_7, "SampleFile.txt - Notepad");
```

Note: The CtxSetWindowTitle function should appear each time the window title changes during capture after the window is created. When the window title contains variable content, such as transaction number or date and time, you can change the CtxSetWindowTitle command to CtxSetWindowMatchTitle and use wildcards for the variable content. To use this approach, you must enable the wildcard matching function using CtxSetEnableWildcardMatching(TRUE).

CtxSetWindowVerification

Enables or disables window verification for actions.

Syntax

void CtxSetWindowVerifiation (BOOL enable);

Return Value

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);

Return Value

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

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 `CtxKeyDown(key)` and `CtxKeyUp(Key)`.

Syntax

void CtxTypeChar(const CtxWI *wi, long vkey, CtxKeyModifierEnum mod);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
key	Virtual Key code to type.
mod	*CtxKeyModifierEnum* The following modes are available: Value — Description NONE — No keyboard modifier was specified SHIFT — Shift key CONTROL — Control key ALT — Alt key

	EXTENDED	An extended key

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 `CtxKeyDown(key)` and `CtxKeyUp(Key)`.

Syntax

void CtxTypeVK(const CtxWI *wi, long vkey, CtxKeyModifierEnum mod);

Return Value

Parameters

Parameter	Description
`wi`	Pointer to a Citrix Window Information object containing window data.
`key`	Virtual Key code to type.
`mod`	*CtxKeyModifierEnum* The following modes are available: Value — Description `NONE` — No keyboard modifier was specified `SHIFT` — Shift key `CONTROL` — Control key `ALT` — Alt key `EXTENDED` — An extended key

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
newcaption	String representing new window caption.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);
```

CtxWaitForFullBitmap

Waits for a full screen bitmap to match the hash code passed into the BitmapHash argument.

Versions

Versions of CtxWaitForFullBitmap are:

void CtxWaitForFullBitmap(char *BitmapHash, char *BitmapTitle, long lTimeoutOffset = 0)

void CtxWaitForFullBitmap(char *BitmapHash, char *BitmapTitle)

CtxWaitForPartialBitmap

Waits for a partial screen bitmap to match the hash code passed into the BitmapHash argument.

Versions

Versions of CtxWaitForPartialBitmap are:

void CtxWaitForPartialBitmap(char *BitmapHash, char *BitmapTitle, int x, int y, int width, int height, long lTimeoutOffset = 0)

void CtxWaitForPartialBitmap(char *BitmapHash, char *BitmapTitle, int x, int y, int width, int height)

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);

Return Value

Parameters

Parameter	Description
x	X coordinate
y	Y coordinate
w	Width of the screen update
h	Height of the screen update
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWindowTimeout.

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWindowTimeout.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowDestroy(CWI_40034, 500);
```

CtxWaitForWindowLgIconChange

Waits for the specified window's large icon to be changed.

Syntax

void CtxWaitForWindowLgIconChange(const CtxWI *wi, const char *hash, long nmWait);

Return Value

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*hash	Unique hash of icon bitmap.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data
x	X coordinate
y	Y coordinate
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

Parameters

Parameter	Description

wi	Pointer to a Citrix Window Information object containing window data.
w	X coordinate.
h	Y coordinate.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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 small icon to be changed.

Syntax

```
void CtxWaitForWindowSmIconChange(const CtxWI *wi, const char *hash, long nmWait);
```

Return Value

Parameters

Parameter	Description
*wi	Pointer to a Citrix Window Information object containing window data.
*hash	Unique hash of icon bitmap.
nmWait	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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);

Return Value

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`	Timeout offset, in milliseconds. Amount to increase or decrease the wait time specified by CtxSetWaitPointTimeout.

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 screen event has already occurred and, if not, waits for the specified time for the event to occur.

Syntax

BOOL CtxWindowEventExists(CitrixWindowEventTypeEnum EventType, int nmWait, const CtxWI *wi);

Return Value

Parameters

Parameter	Description
`EventType`	*CitrixWindowEventTypeEnum* Citrix window event. Valid values are: Value — Description `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();
```

ODBC

ODBC Commands

DO_FreeODBC
Releases the memory used by QALoad's ODBC 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_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_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_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 occurance 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_SQLParamData
Used in conjunction with DO_SQLPutData to supply parameter data at statement execution time.

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 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 does.

DO_FreeODBC

Releases the memory used by *QA*Load 's ODBC driver. It should only be called once at the end of a script.

Syntax

DO_FreeODBC(PLAYERINFO* sInfo);

Return Value

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( int nVersion, PLAYER_INFO* sInfo );
```

Return Value

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, *QA*Load 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_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( int nStmtIndex, int nParamNum, char* sData, int nBufLen );
```

Return Value

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 a 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 the transaction loop or outside the transaction loop.

Use DO_SQLAllocHandle in place of DO_SQLAllocConnect if using ODBC version 3 or higher.

Syntax

```
DO_SQLAllocConnect( int HDBCIndex );
```

Return Value

Parameters

Parameter	Description
ConnectionIndex	Points to a structure that ODBC uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloaddb2", "sa", "" );
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 .

Syntax

```
DO_SQLAllocHandle ( ODBCSQLHandleTypeEnum handleType, int IncomingIndex, int OutgoingIndex)
```

Return Value

Parameters

Parameter	Description
handleType	*ODBCSQLHandleTypeEnum* The type of handle to be allocated. Each handle takes different arguments. Valid values are: Value — Description SQL_HANDLE_ENV — Environment handle SQL_HANDLE_DBC — Connection handle SQL_HANDLE_STMT — Statement handle

	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( int nConnectionIndex, int nStatementIndex );
```

Return Value

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, ODBCSQLCDataTypeEnum CDataType, long
BufferLength, long pBufferLength );
```

Return Value

Parameters

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>StatementIndex</td><td>Index into the table of ODBC statement handles.</td></tr>
<tr><td>ColumnNum</td><td>Number of the result set column to bind.</td></tr>
<tr><td>CDataType</td><td>ODBCSQLCDataTypeEnum
C data type. Valid values are:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SQL_C_BIT</td><td>Bit</td></tr>
<tr><td>SQL_C_UTINYINT</td><td>Unsigned tiny integer</td></tr>
<tr><td>SQL_C_STINYINT</td><td>Signed tiny integer</td></tr>
<tr><td>SQL_C_TINYINT</td><td>Tiny integer</td></tr>
<tr><td>SQL_C_SSHORT</td><td>Signed Short</td></tr>
<tr><td>SQL_C_USHORT</td><td>Unsigned short</td></tr>
<tr><td>SQL_C_SHORT</td><td>Short</td></tr>
<tr><td>SQL_C_SLONG</td><td>Signed long</td></tr>
<tr><td>SQL_C_ULONG</td><td>Unsigned long</td></tr>
<tr><td>SQL_C_LONG</td><td>Long</td></tr>
<tr><td>SQL_C_CHAR</td><td>Char</td></tr>
<tr><td>SQL_C_BINARY</td><td>Binary</td></tr>
<tr><td>SQL_C_FLOAT</td><td>Float</td></tr>
<tr><td>SQL_C_DOUBLE</td><td>Double</td></tr>
<tr><td>SQL_C_DATE</td><td>Date YYYY:MM:DD (for example: 1996:10:25)</td></tr>
<tr><td>SQL_C_TIME</td><td>Time HH:MM:SS (for example: 17:28:01)</td></tr>
<tr><td>SQL_C_TIMESTAMP</td><td>Timestamp YYYY:MM:DD:HH:MM:SS</td></tr>
</table>
</td></tr>
<tr><td>BufferLength</td><td>The length of the buffer for the data that is being returned.</td></tr>
<tr><td>PBufferLength</td><td>The length/indicator buffer to bind to the column.</td></tr>
</table>

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_UINTEGER ); // 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_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,
ODBCSQLBindParameterParamTypeEnum ParamType, ODBCSQLBindParameterCDataTypeEnum CDataType,
ODBCSQLBindParameterSQLDataTypeEnum DataType, long ColumnDefinition, short Scale, SDWORD
InputString, int cbValueMax );
```

Return Value

Parameters

Parameter	Description
`CommandIndex`	Index into the table of ODBC command handles.
nParamNum	Bind parameter number. The first parameter is 1.
`ParamType`	*ODBCSQLBindParameterParamTypeEnum* Defines the direction of the parameter. Valid values are: Value Description

	SQL_PARAM_INPUT	Parameter is input only.
	SQL_PARAM_INPUT_OUTPUT	Parameter is output only.
	SQL_PARAM_OUTPUT	Parameter is input and output.
CDataType	*ODBCSQLBindParameterCDataTypeEnum* C data type. Valid values are:	
	Value	Description
	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_SHORT	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 YYYY:MM:DD (for example: 1996:10:25)
	SQL_C_TIME	Time HH:MM:SS (for example: 17:28:01)
	SQL_C_TIMESTAMP	Timestamp YYYY:MM:DD:HH:MM:SS
DataType	*ODBCSQLBindParameterSQLDataTypeEnum* ODBC SQL data type. Valid values are:	
	Value	Description
	SQL_CHAR	Char
	SQL_DECIMAL	Decimal

	SQL_SMALLINT	Small integer
	SQL_REAL	Real
	SQL_VARCHAR	Varchar
	SQL_TIME	Time HH:MM:SS (for example: 17:28:01)
	SQL_LONGVARCHAR	Long Varchar
	SQL_VARBINARY	Var binary
	SQL_BIGINT	Big integer
	SQL_BIT	Bit
	SQL_NUMERIC	Numeric
	SQL_INTEGER	Integer
	SQL_FLOAT	Float
	SQL_DOUBLE	Double
	SQL_BINARY	Binary
	SQL_LONGVARBINARY	Long variable binary
	SQL_TINYINT	Tiny integer
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 string that defines the input data to the bind call.	
cbValueMax	The length of the output variable being passed.	

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_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( int 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( int StatementIndex )
```

Return Value

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( int StatementIndex, int ColumnNum, SQLUSMALLINT ColumnAttribute,
SQLSMALLINT BufferLen )
```

Return Value

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( int StatementIndex, UCHAR* QualifierName, UCHAR* TableOwner, UCHAR*
TableName, UCHAR* 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( int ConnectionIndex, UCHAR* DSN, UCHAR* UDI, UCHAR* AuthStr );
```

Return Value

Parameters

Parameter	Description
ConnectionIndex	Index into the table of connections.
DSN	Data source name.
UID	User identifier.
AuthStr	Authorization string (password).

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloaddb2", "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( int SourceDescriptorHandleIndex, int TargetDescriptorHandle );
```

Return Value

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, 4291980l, 1242388, 1242384 );
DO_SQLSetDescRec( D1, 2, 4, 0, 4, 10, 0, 4292220l, 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( int StatementIndex, int ColumnNumber, int BufferLength )
```

Return Value

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( int ConnectionIndex );
```

Return Value

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( int ConnectionIndex, UCHAR* ConnectionString );
```

Return Value

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( ODBCSQLTransactionHandleTypeEnum nHandleType, long nHandleIndex,
ODBCSQLTransactTypeEnum nOperation );
```

Return Value

Parameters

Parameter	Description
nHandleType	*ODBCSQLTransactionHandleTypeEnum* The type of Handle on which the transaction is being committed or rolled back. Valid values are: Value — Description SQL_HANDLE_ENV — Environment handle SQL_HANDLE_DBC — Connection handle
nHandleIndex	Index into the table of statement or connection handles, or -666 which is used as a marker for the Environment handle.
nOperation	*ODBCSQLTransactTypeEnum* SQL transaction type option. Valid values are: Value — Description SQL_COMMIT — Commit transaction SQL_ROLLBACK — Rollback transaction

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( int StatementIndex, UCHAR* SQLStatement );
```

Return Value

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( int StatementIndex );
```

Return Value

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 );
DO_LoadMem( S0, 5, "1237", 4 );
DO_SQLExecute( S0 );
```

DO_SQLFetch

Retrieves a single row of data.

Syntax

```
DO_SQLFetch( int StatementIndex )
```

Return Value

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 )
{
DO_SQLGetData( S0, 1, SQL_C_LONG, 4 );
strcpy(pReturnValue, GetFetchedData());
}

DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLFreeConnect

Performs the cleanup of connection handles for ODBC 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( int ConnectionIndex );
```

Return Value

Parameters

Parameter	Description
ConnectionIndex	Points to a structure that ODBC uses to track different connection settings, statements within the connection, and descriptors allocated within the connection.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloaddb2", "sa", "" );
DO_SQLDisconnect( C0 );
DO_SQLFreeConnect( C0 );
```

DO_SQLFreeHandle

In ODBC, DO_SQLFreeHandle handles statement and descriptor cleanup.

Each occurance 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( ODBCSQLHandleTypeEnum HandleType, int HandleIndex )
```

Return Value

Parameters

Parameter	Description
`HandleType`	*ODBCSQLHandleTypeEnum* The type of handle to be allocated. Each handle takes different arguments. Valid values are: Value — Description `SQL_HANDLE_ENV` — Environment handle `SQL_HANDLE_DBC` — Connection handle `SQL_HANDLE_STMT` — Statement handle `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( int StatementIndex, ODBCSQLFreeStmtOptionEnum Option );
```

Return Value

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
Option	*ODBCSQLFreeStmtOptionEnum* SQLFreeStmt option. Valid values are: Value — Description SQL_CLOSE — Close the statement handle SQL_DROP — Drop the statement handle SQL_UNBIND — Unbind the statement binds SQL_RESET_PARAMS — Reset the statement parameters

Example

```
DO_SQLFreeStmt( S0, SQL_DROP );
```

DO_SQLGetCursorName

Use on an open ODBC 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 ( int StatementIndex, short nBufferLength);
```

Return Value

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( int nStmtIndex, int nColNum, int nCType, long 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( int DescriptorIndex, short RecordNumber, short FieldID, long
BufferLength, )
```

Return Value

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( int nDescIndex, SQLSMALLINT nRecordNumber, SQLSMALLINT 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, 4292220l, 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( SQLINTEGER nAttribute, SQLPOINTER strAttrValue, SQLINTEGER nBufferLength,
SQLINTEGER* nStringLength );
```

Return Value

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( int StatementIndex, ODBCSQLGetTypeSQLTypeEnum 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

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>StatementIndex</td><td>Index into the table of statement handles.</td></tr>
<tr><td>SQLType</td><td>ODBCSQLGetTypeSQLTypeEnum
ODBC SQL data type. Valid values are:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SQL_CHAR</td><td>Char</td></tr>
<tr><td>SQL_DECIMAL</td><td>Decimal</td></tr>
<tr><td>SQL_SMALLINT</td><td>Small integer</td></tr>
<tr><td>SQL_REAL</td><td>Real</td></tr>
<tr><td>SQL_VARCHAR</td><td>Varchar</td></tr>
<tr><td>SQL_TIME</td><td>Time HH:MM:SS (for example: 17:28:01)</td></tr>
<tr><td>SQL_LONGVARCHAR</td><td>Long Varchar</td></tr>
<tr><td>SQL_VARBINARY</td><td>Var binary</td></tr>
</table>
</td></tr>
</table>

	SQL_BIGINT	Big integer
	SQL_BIT	Bit
	SQL_NUMERIC	Numeric
	SQL_INTEGER	Integer
	SQL_FLOAT	Float
	SQL_DOUBLE	Double
	SQL_BINARY	Binary
	SQL_LONGVARBINARY	Long variable binary
	SQL_TINYINT	Tiny integer
	SQL_ALL_TYPES	All SQL data types.

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( int StatementIndex );
```

Return Value

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.

Example

```
int pcol = DO_SQLNumResultCols( S1 );
```

DO_SQLParamData

Used in conjunction with DO_SQLPutData to supply parameter data at statement execution time.

Syntax

int DO_SQLParamData(int nStatementIndex);

Parameters

Parameter	Description
nStatementIndex	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_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( int StatementIndex, UCHAR* SQLString );
```

Return Value

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_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);
```

Return Value

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_SQLRetrieveParamValue

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( int nStmtIndex, short nParamNumber );
```

Return Value

Parameters

Parameter	Description

nStmtIndex	Index into the table of ODBC 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 );
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( int nStatementIndex );
```

Return Value

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( int nConnectionIndex, long nAttribute, void* nAttrValue, long
nStrLength );
```

Return Value

Parameters

Parameter	Description
nConnectionIndex	Points to a structure that ODBC 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 *QA*Load 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.

Example

```
DO_SQLAllocConnect( C0 );
DO_SQLConnect( C0, "fhloaddb2", "sa", "" );
DO_SQLAllocHandle( SQL_HANDLE_DBC, 0, C1 );
DO_SQLSetConnectAttr( C1, SQL_ATTR_AUTOCOMMIT, "SQL_AUTOCOMMIT_OFF", SQL_NTS );
```

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( int ConnectionIndex, ODBCSQLSetConnectOptionEnum Option, UDWORD Value );
```

Return Value

Parameters

Parameter	Description
ConnectionIndex	Index into a table of ODBC connection handles.
Option	*ODBCSQLSetConnectOptionEnum* One of the valid option constants. Valid values are: Value — Description SQL_ACCESS_MODE — Determines type of access this program uses SQL_AUTOCOMMIT — 0 = Autocommit off, 1 = Autocommit on SQL_LOGIN_TIMEOUT — Number of seconds to wait for a login request to complete before returning to the application. The default is 15 SQL_OPT_TRACE — Integer value telling the Driver Manager whether or not to perform tracing. 0 = Tracing off (Default) 1 = Tracing on SQL_OPT_TRACEFILE — 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	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	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	32-bit bitmask that sets the transaction isolation level for the current connection index. Refer to ODBC documentation for details on setting this parameter
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( int StatementIndex, UCHAR* CursorName );
```

Return Value

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( int DescriptorHandle, short RecordNumber, short FieldID, SQLPOINTER
Value, long BufferLen )
```

Return Value

None

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( int nDescIndex, short nRecordNumber, short nField, short nSubType, long nLength, short nPrecision, short nScale, char* pData, long pStrLen, long pIndicator )
```

Return Value

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.

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.

Environment connection elements are set automatically in DO_initODBC.

Syntax

```
DO_SQLSetEnvAttr( SQLINTEGER Attribute, void* AttributeValue, SQLINTEGER Indicator)
```

Return Value

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.

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( int StatementIndex, UWORD nRow, UWORD nRefresh, UWORD nLock );
```

Return Value

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_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 );
```

Return Value

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, *QA*Load 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	A 32-bit value that sets the binding orientation used when DO_SQLExtendedFetch is called on the associated C. Column-wise binding is selected by supplying the defined constant SQL_BIND_BY_COLUMN for the argument vParam. Row-wise binding is selected by supplying a value for vParam specifying the length of a structure or an instance of a buffer into which result columns will be bound. The length specified in vParam 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. Column-wise binding is the default binding orientation for DO_SQLExtendedFetch.
SQLMaxLength	A value corresponding to the maximum amount of data that can be retrieved from a single column with a LONG VARCHAR or LONG VARBINARY data type.
SQLMaxRows	A value corresponding to the maximum number of rows to return to the application for a SELECT command. If vParam equals 0 (Default), the driver returns all rows.
SQLNoScan	A value. If TRUE (1), the driver does not scan SQL strings for escape clauses. Instead, the driver sends the command directly to the data source. If FALSE (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 vParam equals 0 (Default), there is no time out.

Syntax

```
DO_SQLSetStmtOption( int StatementIndex, ODBCSQLSetStmtOptionEnum nOption, UDWORD nParam );
```

Return Value

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
nOption	*ODBCSQLSetStmtOptionEnum* One of the valid option constants. Valid values are: Value — Description SQL_QUERY_TIMEOUT — An integer value corresponding to the number of seconds to wait for an SQL statement to execute before returning to the application. If vParam equals 0 (Default), there is no time out SQL_MAX_ROWS — An integer value corresponding to the maximum number of rows to return to the application for a

		SELECT command. If vParam equals 0 (Default), the driver returns all rows
	SQL_NOSCAN	An integer value. If TRUE (1), the driver does not scan SQL strings for escape clauses. Instead, the driver sends the command directly to the data source. If FALSE (Default, value 0), the driver scans for escape clauses
	SQL_MAX_LENGTH	An integer value corresponding to the maximum amount of data that can be retrieved from a single column with a LONG VARCHAR or LONG VARBINARY data type
	SQL_ASYNC_ENABLE	A 32-bit integer value that specifies whether a function called with the specified hstmt is executed asynchronously@ SQL_ASYNC_ENABLE_OFF = Off (Default), SQL_ASYNC_ENABLE_ON = On
	SQL_BIND_TYPE	A 32-bit integer value that sets the binding orientation used when DO_SQLExtendedFetch is called on the associated C. Column-wise binding is selected by supplying the defined constant SQL_BIND_BY_COLUMN for the argument vParam. Row-wise binding is selected by supplying a value for vParam specifying the length of a structure or an instance of a buffer into which result columns will be bound
	SQL_CURSOR_TYPE	A 32-bit integer value that specifies the cursor type@SQL_CURSOR_FORWARD_ONLY = The cursor only scrolls forward. SQL_CURSOR_STATIC = The data in the result set is static. SQL_CURSOR_KEYSET_DRIVEN = The driver saves and uses the keys for the number of rows specified in the SQL_KEYSET_SIZE statement option. SQL_CURSOR_DYNAMIC = The driver only saves and uses the keys for the rows in the rowset. The default value is SQL_CURSOR_FORWARD_ONLY. This option cannot be specified for an open cursor and can also be set through the crowKeyset argument in SQLSetScrollOptions
	SQL_CONCURRENCY	A 32-bit integer value that specifies the cursor concurrency@SQL_CONCUR_READ_ONLY = Cursor is read-only. No updates are allowed. SQL_CONCUR_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated. SQL_CONCUR_ROWVER = Cursor uses optimistic concurrency control, comparing row versions, such as SQLBase ROWID or Sybase TIMESTAMP. SQL_CONCUR_VALUES = Cursor uses optimistic concurrency control, comparing values. The default value is

		SQL_CONCUR_READ_ONLY
	SQL_KEYSET_SIZE	A 32-bit integer value that specifies the number of rows in the keyset for a keyset-driven cursor. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside of the keyset). The default keyset size is 0
	SQL_ROWSET_SIZE	A 32-bit integer value that specifies the number of rows in the rowset. This is the number of rows returned by each call to SQLExtendedFetch. The default value is 1
	SQL_SIMULATE_CURSOR	A 32-bit integer value that specifies whether drivers that simulate positioned update and delete statements guarantee that such statements affect only one single row
	SQL_RETRIEVE_DATA	A 32-bit integer value@SQL_RD_ON = SQLExtendedFetch retrieves data after it positions the cursor to the specified location. This is the default. SQL_RD_OFF = SQLExtendedFetch does not retrieve data after it positions the cursor
	SQL_USE_BOOKMARKS	A 32-bit integer value that specifies whether an application will use bookmarks with a cursor@ SQL_UB_OFF = Off (Default), SQL_UB_ON = On
nParam	The parameter value.	

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( int StatementIndex, ODBCSQLSpecialColumnsColTypeEnum fColType, UCHAR*
szTableQualifier, UCHAR* szTableOwner, UCHAR* szTableName, ODBCSQLSpecialColumnsScopeEnum
fScope, ODBCSQLSpecialColumnsNullableEnum fNullable );
```

Return Value

Parameters

Parameter	Description
StatementIndex	Index into the table of statement handles.
fColType	*ODBCSQLSpecialColumnsColTypeEnum* Type of column to return. Valid values are: Value — Description 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	*ODBCSQLSpecialColumnsScopeEnum* Minimum required scope of the ROWID. The returned ROWID may be of greater scope. Valid values are: Value — Description 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	*ODBCSQLSpecialColumnsNullableEnum*

	values are: Value — Description 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( int StatementIndex, UCHAR* szTableQualifier, UCHAR* szTableOwner, UCHAR*
szTableName, ODBCSQLStatisticsUniqueEnum fUnique, ODBCSQLStatisticsAccuracyEnum fAccuracy );
```

Return Value

Parameter

Parameter	Description
StatementIndex	Index into the table of statement handles.
SzTableQualifier	Qualifier name.
SzTableOwner	Owner name.
SzTableName	Table name.
fUnique	*ODBCSQLStatisticsUniqueEnum* Type of index. Valid values are: Value — Description SQL_INDEX_UNIQUE — Unique value index SQL_INDEX_ALL — Non-unique value index
fAccuracy	*ODBCSQLStatisticsAccuracyEnum* Importance of the CARDINALITY and PAGES columns in result set. Valid values are:

	Value	Description
	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( int StatementIndex, UCHAR* szTableQualifier, UCHAR* szTableOwner, UCHAR*
szTableName, UCHAR* szTableType );
```

Return Value

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( int ConnectionIndex, ODBCSQLTransactTypeEnum fType );
```

Return Value

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	*ODBCSQLTransactTypeEnum* SQL transaction type option. Valid values are: Value — Description SQL_COMMIT — Commit transaction SQL_ROLLBACK — Rollback transaction

Example

```
DO_SQLFreeStmt( C1, SQL_DROP );
DO_SQLTransact( S0, SQL_COMMIT );
```

DO_substr

Finds a value within a string.

Syntax

```
DO_Substr( char* string, int placeholder, char* value );
```

Return 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);
```

Return Value

char* containing the data or an error

Parameters

Parameter	Description
nIndex	The statement index.
nColumn	The column of data to return.
nRow	The row of data to return.

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 Forms Server

Oracle Forms Server Commands

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.

ofsCollapseTreeItem
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.

ofsDeIconifyWindow
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.

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.

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. In Servlet mode and Secure Servlet mode, also returns the HTTP headers.

ofsHideWindow
Adds the Visible property (with Disabled attribute) to the current message.

ofsHTTPDisconnect
Closes the current HTTP connection to the Forms Listener servlet.

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.

ofsWindowCreated
Check if the specified window was created during the last transaction with the server (ofsSendRecv).

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, OFSActionTypeEnum
ActionType, int PropertyID, const char *sValue );
```

Return Value

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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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 );
:
:
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, OFSActionTypeEnum
ActionType, int PropertyID, const char *sValue);
```

Return Value

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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message.

	OFS_ENDMSG	Add the 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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message.

	OFS_ENDMSG	Add the 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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the property to the current message and end 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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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 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, OFSActionTypeEnum
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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, 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, OFSActionTypeEnum ActionType, int
PropertyID, const char *Value);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return Value

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 property, with Disabled attribute, indicates the ending of a currently opened window.

Syntax

```
void ofsDeActivateWindow(const char *sHandlerName, int ControlID, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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_DATABASETEST"
//being terminated prior to the ending of the HTTP session.

ofsDeActivateWindow( "WINDOW_DATABASETEST", 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, OFSActionTypeEnum
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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum
ActionType, int PropertyID, const char *Value);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum
ActionType, int PropertyID, const char *Value);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return 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, OFSActionTypeEnum
ActionType, int PropertyID, const char *Value);
```

Return 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	*OFSActionTypeEnum*

	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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum ActionType,
int PropertyID, const char *Value);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are:

	Value	Description
	OFS_ADD	Add the property to the current message.
	OFS_ENDMSG	Add the 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);
```

Return 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.

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 );
```

ofsGetServerData

Returns the Forms data from the server reply. In Servlet mode and Secure Servlet mode, also returns the HTTP headers.

The statement is manually added to the script in conjunction with DO_ExtractString, which extracts a substring from the returned data. The extracted value is passed to subsequent script statements.

The combination of DO_ExtractString and ofsGetServerData statements enables you to obtain and use dynamic Forms data, based on the server response. These statements replace the functionality provided by the OFS Java script statement GetControlValue.

Syntax

```
const char *ofsGetServerData();
```

Return Value

Returns the Forms data from the server reply. In Servlet mode and Secure Servlet mode, also returns the HTTP headers.

Parameters

None

Example

The example below executes RR__printf and DO_ExtractString statements after the ofsSendRecv statement.

Both statements use the ofsGetServerData statement as a parameter. After printing the result value, the memory allocated by DO_ExtractString() is freed, and the pointer is set to NULL.

> Caution: The value returned by the ofsGetServerData command should not be freed. Calling free on this pointer results in a memory error in the script.

```
.....
/* Declare Variables */
char *szResult = NULL;
...
BEGIN_TRANSACTION();
...
...
ofsSendRecv(1); //ClientSeqNo=1|CapTime=1090942125.437|1090942125.437
DO_ExtractString( ofsGetServerData(), /* returns character string containing Forms data */
1,
"P|S|284|java.lang.Integer|0|", /*left filter param*/
"|P|S|284|java.lang.Integer|0|657930", /*right filter param*/
&szResult /* Value contains the dynamic value to be used in subsequent statements */
);
RR__printf("item: %s", szResult);
free(szResult);
szResult = NULL;
```

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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return Value

Parameters

Parameter	Description
sformsServletURL	The URL location of the Forms Servlet.

Example

```
ofsHTTPConnectToFormsServlet( "http://ntsap45b:7779/forms90/f90servlet?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);
```

Return Value

Parameters

Parameter	Description
sformsServletURL	The URL location of the Forms Listener servlet.

Example

```
ofsHTTPConnectToListenerServlet( "http://ntsap45b:7779/forms90/l90servlet");
```

ofsHTTPDisconnect

Closes the current HTTP connection to the Forms Listener Servlet.

Syntax

```
void ofsHTTPDisconnect();
```

Return Value

Parameters

None

Example

```
ofsHTTPDisconnect();
```

ofsHTTPInitialFormsConnect

Opens an HTTP connection to the Forms Listener servlet and posts the initial Forms handshake information.

Syntax

```
void ofsHTTPInitialFormsConnect();
```

Return Value

Parameters

None

Example

```
ofsHTTPInitialFormsConnect();
```

ofsHTTPSetHdrProperty

Establishes the HTTP headers to use for connecting to the Forms servlet and listener servlet. Headers that can be set with this function are: Cookie, User-Agent, Host, Accept, and Connection.

Syntax

```
void ofsHTTPSetHdrProperty(const char *sName, const char *sValue);
```

Return Value

Parameters

Parameter	Description
sName	The HTTP header name.
sValue	The HTTP header value.

Example

```
ofsHTTPSetHdrProperty("User-Agent", "Java1.3.1.9" );
ofsHTTPSetHdrProperty("Host", "ntsap45b.prodti.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);
```

Return Value

Parameters

Parameter	Description
sListenerServlet	Servlet parameters to use for this session.

Example

```
ofsHTTPSetListenerServletParms( "?ifcmd=getinfo&ifhost=C104444D01&ifip="
"192.168.234.1" );
```

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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return 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.
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.
iCoordianteX	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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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. This corresponds to the row in the list of values.

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);
```

Return Value

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);
```

Return Value

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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum*

	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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return Value

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);
```

Return Value

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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return Value

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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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();
```

Return Value

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);
```

Return Value

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);
```

Return Value

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 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);
```

Return Value

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 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);
```

Return Value

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.

This is a keyword search. You can broaden the search to include more than one specific transaction by modifying the Expected Server Message parameter to look for part of a string.

Syntax

```
void ofsSetExpectedServerMsg(const char *ExpectedServerMessage);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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 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);
```

Return Value

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 and WWW Universal Sessions in OFS Advanced Scripting Techniques.

Note: The memory buffer allocated by ofsSetICXTicket needs to be explicitly freed.

Syntax

```
ofsSetICXTicket( char **cookieValue);
```

Return Value

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 rquest 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);
   // Free the memory allocated by DO_GetUniqueString and ofsSetICXTicket before the end of
the transaction.
      free(p);
      p=NULL;
      free(pTicket);
      pTicket=NULL;
END_TRANSACTION()
```

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, OFSActionTypeEnum iAction,
int iPropertyID, OFSFormsVersionEnum sFormsVersion);
```

Return Value

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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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, represented as an integer string.

Example

```
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "101200" );
```

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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);

.
.
.

ofsSetNoRequiredVAList( "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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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, OFSActionTypeEnum
ActionType, int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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( OFSFormsVersionEnum sFormsVersion, OFSConnectionTypeEnum
iConnectType, int iHeartbeatInterval, OFSMessageCheckingEnum iCheckServerMsgs);
```

Return Value

Parameters

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>sFormsVersion</td><td>OFSFormsVersionEnum
Version of Oracle Forms in use. Valid values are:

Value — Description
FORMS_10g_SERVLET — Oracle Application Server 10g
FORMS_9i_SERVLET — Oracle Application Server 9i
FORMS_6i_SERVLET — Oracle Forms 6i Servlet Mode
FORMS_6i_11i_SERVLET — Oracle Forms 6i in 11i Environment
FORMS_6i_SOCKET — Oracle Forms 6i Socket Mode
FORMS_60_SOCKET — Oracle Forms 6.0</td></tr>
<tr><td>iConnectType</td><td>OFSConnectionTypeEnum
This flag indicates the type of connection to establish with the server. Valid values are:</td></tr>
</table>

	Value / Description OFS_SOCKET — Socket connection OFS_HTTP — HTTP connection OFS_HTTPS — Secure (SSL) connection
iHearbeatInterval	Heartbeat interval (minutes).
iCheckServerMsgs	*OFSMessageCheckingEnum* 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. Valid values are: Value / Description OFS_DONTCHECKMSGS — Disable error message checking for server messages. OFS_CHECKMSGS — Enable error message checking for server messages.

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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, OFSServerMessageComparisonTypeEnum
iMsgOption);
```

Return Value

Parameters

Parameter	Description
sMsgString	String to compare against server message
iMsgOption	*OFSServerMessageComparisonTypeEnum* This flag indicates the string comparison method to use for comparing a string against a server message. Valid values are: Value — Description 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);
```

Return Value

Parameters

Parameter	Description
iConnectMode	Connection mode.
sServletName	The listener servlet name.

Example

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

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);
```

Return Value

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 );
```

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);
```

Return Value

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");
```

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);
```

Return Value

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, OFSActionTypeEnum ActionType,
int PropertyID);
```

Return 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	*OFSActionTypeEnum* 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. Valid values are: Value — Description OFS_ADD — Add the property to the current message. OFS_ENDMSG — Add the 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();
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

Return Value

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);
```

ofsWindowCreated

See also Oracle Forms Server

Check if the specified window was created during the last transaction with the server (ofsSendRecv).

Syntax

```
int ofsWindowCreated(int iControlID, char *sControlName);
```

Return Value

TRUE if the window was created, FALSE otherwise

Parameters

Parameter	Description
iControlID	Captured ID of the GUI control for the current message.
sControlName	Name of the control.

Example

```
ofsSendRecv(1);
ofsWindowCreated(710, "Account information");
```

QALoad

QALoad Common Commands

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_MSLEEP
Inserts a sleep for the number of seconds defined in the parameter.

DO_SetTransactionCleanup
Defines a point at the end of the transaction for anything that needs to be de-allocated 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_DATAPOOLS_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.

Modify_Encoding
Modifies the encoding for a string parameter.

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.

OPEN_SHARED_DATA_POOL
Opens the shared datapool file.

RANDOM_NUMBER
Returns a string representation of a random number.

RANDOM_STRING
Returns a string with a 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.

SET_SCRIPT_LANGUAGE
Specifies the encoding used for literal strings contained within the script. The default encoding is "SLID_English".

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, 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( );
```

Return Value

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 ( char* CheckpointName );
```

Return Value

Parameters

Parameter	Description
CheckpointName	String containing a description of the checkpoint. This value cannot be longer than 127 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 ();
```

Return Value

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);
```

Return Value

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.

Versions

Versions of COUNTER_VALUE are:

COUNTER_VALUE (int Counter_ID, long Counter_Value);

COUNTER_VALUE (int Counter_ID, float Counter_Value);

DATE_TIME

Gets the current time and/or date from the local machine.

Syntax

```
char * DATE_TIME(const char *pformat);
```

Return Value

char *: Formatted date/time string. This string should be freed when it is no longer needed.

Parameters

Parameter	Description
pformat	A formatted control string that determines how to format the date/time string. The following formats can be used: ! %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);
```

Return Value

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, CounterDataTypeEnum
Data_Type, CounterCounterTypeEnum Counter_Type );
```

Return Value

any value except -1 if successful
-1 if unsuccessful

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	*CounterDataTypeEnum* Counter data type. Valid values are: Value Description

	DATA_LONG	The counter values will be of type long.
	DATA_FLOAT	The counter values will be of type float.
Counter_Type	*CounterCounterTypeEnum* Counter type. Valid values are: Value — Description COUNTER_CUMULATIVE — The counter data is cumulative. COUNTER_INSTANCE — The counter data is 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 ( char* text );
```

Return Value

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( bool flag );
```

Return Value

Parameters

Parameter	Description
flag	TRUE or FALSE. A flag indicating whether the script should abort upon receiving an error. 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.

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.

Note: The left string and the right string must be provided or an error is returned.

Syntax

```
BOOL DO_ExtractString(const char* szBuffer, int nCount, const char* szLeft, const char* szRight, char** pszResult);
```

Return Value

TRUE if successfulchar* szResult = NULL;

...

/*

* The page returns a page containing "<title>Enter Login</title>"

*/

DO_Http("GET http://www.host.com/ HTTP/1.1\ r\ n\ r\ n");

DO_ExtractString(DO_GetReplyBuffer(), 1, "<title>", "</title>", & szResult);

/*

* prints "The extracted title: Enter Login"

*/

RR__printf("The extracted title: %s", szResult);

free(szResult);

szResult = NULL;

FALSE if not successful

DO_ExtractString allocates enough space in the parameter passed in as the string buffer to hold the string (including the NULL).

Caution: The result parameter variable should be explicitly initialized to NULL. Failure to do so results in a memory error in the script.
Once the result parameter has been allocated, it can be reused within the same transaction loop without being explicitly freed. However, the buffer memory should be freed at the end of the transaction loop and the pointer should be set to NULL. Failure to do so results in a memory leak.

Parameters

Parameter	Description
`szBuffer`	The buffer to search.
`nCount`	The number of occurrences in the left string before a match is made.
`szLeft`	The left side of the string to match.
`szRight`	The right side of the string to match.
`pszResult`	The address of the return string.

Example

```
char* szResult = NULL;
...
/*
* The page returns a page containing "<title>Enter Login</title>"
*/
DO_Http("GET http://www.host.com/ HTTP/1.1\r\n\r\n");
DO_ExtractString(DO_GetReplyBuffer(), 1, "<title>", "</title>", &szResult);
/*
* prints "The extracted title: Enter Login"
*/
RR__printf("The extracted title: %s", szResult);
free(szResult);
szResult = NULL;
```

DO_MSLEEP

Inserts a sleep for the number of seconds defined in the parameter.

The parameter passed to DO_MSLEEP 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_MSLEEP 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_MSLEEP command is encountered, it sleeps for a random time frame ranging from 0 to the value specified.

Syntax

```
DO_MSLEEP( int nMilliseconds );
```

Return Value

Parameters

Parameter	Description
nMilliseconds	Number of milliseconds 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_MSLEEP( 5 );  /* Sleep 5 seconds */
```

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() ;
```

Return Value

Parameters

None.

Example

```
BEGIN_TRANSACTION();
DO_SetTransactionStart();
TRANSACTION CODE...
DO_SetTransactionCleanup();
DO_HttpCleanup();
DO_SomeOtherMiddlewareCleanup();
END_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. This function is used in conjunction with DO_SetTransactionCleanup.

Syntax

```
DO_SetTransactionStart() ;
```

Return Value

Parameters

None.

Example

```
BEGIN_TRANSACTION();
DO_SetTransactionStart();

TRANSACTION CODE...
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".

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 )
```

Return Value

TRUE if successful
FALSE if unsuccessful.

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( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	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, *QA*Load 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 ( );
```

Return Value

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 ( char* CheckpointName ) ;
```

Return Value

Parameters

Parameter	Description
CheckpointName	String containing a description of the checkpoint. This value cannot be longer than 127 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 ( );
```

Return Value

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 ();
```

Return Value

int -- absolute virtual user number

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 ();
```

Return Value

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 one (1).

Syntax

```
GET_DATA_FIELD (int datapool ID, int FieldNum);
```

Return Value

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 one (1).

Example

```
BeginCheckpoint();
RR__printf("Datapool Entry #1: %s", GET_DATA_FIELD (1, 1) );
DO_SLEEP(500);
EndCheckpoint(1);
```

GET_DATAPOOLS_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()
```

Return Value

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()
```

Return Value

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()
```

Return Value

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 ();
```

Return Value

int -- relative virtual user number

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()
```

Return Value

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()
```

Return Value

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 );
```

Return Value

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 rhobot_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);
}
```

Modify_Encoding

Modifies the encoding for a string parameter.

Modify_Encoding is used in scripts to convert strings to UTF8, EUCJP or to the language used by the script.

Syntax

```
char* Modify_Encoding(PLAYERINFO* pInfo, EncodingLangEnum encodingID, const char* strInput,
char** szResult)
```

Return Value

A char pointer to the encoded string if successful; NULL if not successful.

Parameters

Parameter	Description
pInfo	Pointer to the PLAYERINFO struct, sinfo.
encodingID	*EncodingTypeEnum* Counter data type. Valid values are: Value — Description UTF8 — UTF8 encoding. EUCJP — EUCJP encoding.

	SCRIPT_LANGUAGE Script language encoding.
strInput	The input string.
szResult	The resulting encoded string. Note that this buffer will need to be freed to prevent memory leaks.

Example

The following is an example of a encoding a string:

```
Modify_Encoding(SCRIPT_LANGUAGE, DO_Http("GET http://www.google.com/ HTTP/1.0\r\n\r\n"),
&test);
```

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);
```

Return Value

n The length of the string after conversion.

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 (char* filename, int poolNumber, unsigned short fileAction);
```

Return Value

TRUE if the file was successfully opened; FALSE otherwise.

Parameters

Parameter	Description
filename	The name of the datapool file, including a drive and path.
poolNumber	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.
fileAction	Action to be taken when the "End of File" is reached. DPA_REWIND_AT_EOF If the datapool file should be rewound to the beginning after it reaches the end. DPA_DONT_REWIND if the datapool file should not be rewound.

Example

```
OPEN_DATA_POOL( "C:\\Program Files\\Compuware\\ QALoad \\Middlewares\\
   SQLServer\\Scripts\\junk.dat", SS_1, DPA_REWIND_AT_EOF );
```

RANDOM_NUMBER

Returns a string representation of a random number between Low and High using Leading and Decimals to format the number.

The seed value that is used to generate the random number is automatically generated from the Player.

Syntax

```
char* RANDOM_NUMBER(int Low, int High, int Leading, int Decimals);
```

Return Value

char*: A pseudo-random random number string. This string should be freed when it is no longer needed.

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"
// "099.03"
// "077.12"
```

RANDOM_STRING

Returns a string with a random set of alpha or alphanumeric characters of the specified width.

The seed value that is used to generate the random number is automatically generated from the Player.

Syntax

```
char* RANDOM_STRING(int AlphaNum, int WidthMin, int WidthMax);
```

Return Value

char*: A pseudo-random random alphanumeric string. This string should be freed when it is no longer needed.

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( int datapool_ID );
```

Return Value

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 ( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	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 *pPlayerInfo, char* msg);
```

Return Value

Parameters

Parameter	Description
pPlayerInfo	Pointer to the PLAYERINFO struct, sinfo.
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 ();
```

Return Value

True if the debug flag is on.

False if the debug flag is off.

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]...);
```

Return Value

The number of characters printed or a negative value if an error occurs.

Parameters

Parameter	Description
format	Format control.
argument	Optional arguments.

Example

```
/* This code segment shows examples of the usage of the RR__printf function to produce
formatted output for various datatypes.  */
```

char ch='h', *string="computer";
int count=-9234;
double fp=251.7366;
wchar_t wch=L'w', *wstring=L"Unicode";

```
/*Display integers. */
RR__printf("Integer formats:\n" "\tDecimal: %d Justified: %.6d Unsigned: %u\n", count,
count, count, count);
```

```
RR__printf("Decimal %d as:\n\tHex: %Xh C hex: 0x%x Octal: %o\n", count, count, count,
count);

/* Display in different radixes. */
RR__printf("Digits 10 equal:\n\tHex: %i Octal: %i Decimal: %i\n",0x10, 010, 10);

/* Display characters. */
RR__printf("Characters in field:\n%10c%5hc%5C%5lc\n", ch, ch, wch, wch);

/* Display strings. */
RR__printf("Strings in field:\n%25s\n%25. 4hs\n\t%S%25.3ls\n", string, string, wstring,
wstring);

/* Display real numbers. */
RR__printf("Real numbers:\n\t%f%.2f%e%E\n", fp, fp, fp, fp);

/* Display pointer. */
RR__printf("\nAddress as:\t%p\n", &count);

/* Count characters printed. */
RR__printf("\nDisplay to here:\n");
RR__printf("1234567890123456%n78901234567890\n", &count);
RR__printf("\tNumber displayed: %d\n\n", count);
```

Output

```
Integer formats:

                    Decimal: -9234
                    Justified: -009234
                    Unsigned: 4294958062

Decimal -9234 as:

                    Hex: FFFFDBEEh
                    C hex: 0xffffdbee
                    Octal: 37777755756

Digits 10 equal:

                    Hex: 16
                    Octal: 8
                    Decimal: 10

Characters in field:

                    h h w w

Strings in field:

                    computer
                    4hs
                    Uni

Real numbers:

                    251.736600
                    251.74 2.517366e+002


Address as:

                    0141FDC0

Display to here:

                    12345678901234567890123456789O

Number displayed:

                    16
```

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 rhobot_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);
}
```

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 ( char* functionName );
```

Return Value

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( ) ;
}
```

SET_SCRIPT_LANGUAGE

Specifies the encoding used for literal strings contained within the script. The default encoding is "SLID_English".

Note: This statement must precede the initialization of the middleware being used.

Syntax

void SET_SCRIPT_LANGUAGE (SCRIPT_LANG languageID);

Return Value

None

Parameters

Parameter	Description
languageID	Identifies the language encoding of strings within the script. Possible values are: ! SLID_English ! SLID_Chinese_Simplified ! SLID_Chinese_Traditional ! SLID_Japanese ! SLID_Korean

Example

```
// Establish 'Japanese' as the language of the encoded strings

SET_SCRIPT_LANGUAGE (SLID_Japanese);

DO_InitHttp(s_info); // SET_SCRIPT_LANGUAGE() must precede the

// middleware's initialization statement
```

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 ( int nSeconds );
```

Return Value

Parameters

Parameter	Description
nSeconds	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( );
```

Return Value

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.

Note: This function applies to virtual users that are already running. SYNCH does not apply to users who have not yet started the test.

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)
```

Return Value

Parameters

Parameter	Description
n	The datapool field number. Field numbering starts at 1.

Example

```
Do_TuxFMLData ( 8302, 0, VARDATA(1));
```

SAP

SAP Commands

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.

SAPGuiContentCheck
Compares data from an SAP server returned control with input string based on the comparison options.

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.

SAPGuiGetControlText

Extracts data from a SAP server returned control.

SAPGuiGetUniqueString
Extracts data from an SAP server returned control that occurs between the left and right input strings.

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.

SAPGuiApplication

Allows scripts to call low-level administrative objects of the SAP GUI.

Syntax

SAPGuiApplication(char* 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 ( const char* OKCode, const char* ScreenName, const char* title );
```

Return Value

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

SAPGuiCmd3(GuiMainWindow,ResizeWorkingPane,94,24,false);
SAPGuiPropIdStr("wnd[0]/tbar[0]/okcd");
SAPGuiCmd1(GuiOkCodeField,PutText,"bibs");
SAPGuiPropIdStr("wnd[0]");
SAPGuiCmd1(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 ( const char* ID, const char* statusBarValue );
```

Return Value

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( const char* Type, const char* FuncName);
```

Return Value

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");
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" );
```

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( const char* Type, const char* FuncName, const char* Param);
```

Return Value

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( const char* Type, const char* FuncName, ISapGenericCollectionPtr Coll, const char* Param);
```

Return Value

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( const char* Type, const char* SubPropType, const char* FuncName,
IDispatchPtr ElmntAry, int ElmntIndx, const char* SubFuncName, const char* Param);
```

Return Value

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( const char* Type, const char* SubPropType, const char* FuncName, const char*
SubFuncName, const char* Param);
```

Return Value

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( const char* Type, const char* SubPropType, const char* FuncName, const char*
SubFuncName, const char* Param, const char* SubParam);
```

Return Value

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( const char* Type, const char* FuncName, const char* Param1, const char* Param2);
```

Return Value

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/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);
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( const char* Type, const char* FuncName, const char* Param1, const char* Param2,
const char* Param3);
```

Return Value

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

```
HRESULT SAPGuiConnect( PLAYER_INFO* s_info, char* server description);
```

Return Value

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");
```

SAPGuiContentCheck

Compares data from an SAP server returned control with input string based on the comparison options.

Syntax

```
int SAPGuiContentCheck(const char* id, const char* type, const char* searchString, Bool bCaseSensitive, SAP_CONTENTCHECK_OPTION nType);
```

Return Value

-1 When an error occurs, for example, the control does not exist.

0 If the content and the input string are identical (nType = ENTIRE)
If the input string is prefix of content (nType = PREFIX)
If the input string is suffix of content (nType = SUFFIX)
If the input string is substring of content (nType = SUBSTRING)

1 If the content and the input string are not identical (nType = ENTIRE)
If the input string is not prefix of content (nType = PREFIX)
If the input string is not suffix of content (nType = SUFFIX)
If the input string is not substring of content (nType = SUBSTRING)

Parameters

Parameter	Description
id	The SAP control ID.
type	The SAP control type.
searchString	A character string specifying a content to search for in the control's text property.
bCaseSensitive	Case sensitive or case insensitive comparison.
nType	*SAP_CONTENTCHECK_OPTION* Type corresponding to the comparison options available on the QALoad Script Development Workbench Convert Options wizard. Valid values are: Value — Description ENTIRE — Compare entire SAP text PREFIX — Compare SAP text prefix SUFFIX — Compare SAP text suffix SUBSTRING — Compare SAP text substring

Example

```
int n;
....
...
n = SAPGuiGetContentCheck("wnd[0]/usr/txtRSYST-BNAME","GuiTextField","qa", false, PREFIX);
RR__printf("return value  = %d", n);
...
..
```

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( char* Type, char* FuncName, ISapGenericCollectionPtr Coll)
```

Return Value

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( const char* Type, ISapGenericCollectionPtrColl );
```

Return Value

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" );
```

SAPGuiGetControlText

Extracts data from a SAP server returned control.

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

Syntax

```
char* SAPGuiGetControlText(const char id, const char* type);
```

Return Value

The string (null-terminated) of characters containing the text of an SAP control.

SAPGuiGetControlText allocates enough space in the parameter passed in as the string buffer to hold the string (including the NULL). Please remember to free any memory after using the returned string. Any memory created with this command that is not explicitly freed results in a memory leak.

Parameters

Parameter	Description
id	The SAP control ID.
type	The SAP control type.

Example

```
char *p;
....
...
p = SAPGuiGetControlText("wnd[0]/usr/txtRSYST-MANDT", "GuiTextField");
RR__printf("text = %s", p);
...
..
free(p);
```

SAPGuiGetUniqueString

Extracts data from an SAP server returned control that occurs between the left and right input strings.

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

Syntax

```
char* SAPGuiGetUniqueString (const char id, const char* type, const char* left, const char* right);
```

Return Value

The string (null-terminated) of characters between the left and right search strings.

NULL if either the left or right search strings are not found, an error message will also be given.

NULL if the left search string and the right search string convert all text property of the SAP control.

SAPGuiGetUniqueString allocates enough space in the parameter passed in as the string buffer to hold the string (including the NULL). Please remember to free any memory after using the returned string. Any memory created with this command that is not explicitly freed results in a memory leak.

Parameters

Parameter	Description
id	The SAP control ID.
type	The SAP control type.
left	A string containing the left search string.

right	A string containing the right search string.

Example

```
char *p;
....
...
p = SAPGuiGetUniqueString("wnd[0]/usr/txtRSYST-BNAME","GuiTextField","qa","23");
RR__printf("String value  = %s", p);
...
..
free(p);
```

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( char* Object_ID );
```

Return Value

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);
```

Return Value

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);
```

Return Value

Parameters

Parameter	Description
Object_Id	String used for matching `SAPGuiPropIdStrExists` 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( const char* FuncName, const char* Param1 )
```

Return Value

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", "SAPLSPO1", "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);
```

Return Value

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( char* version );
```

Return Value

Parameters

Parameter	Description
version	String that includes the major version number, minor version number, and patch level number of the installed SAP client.

	Format: "Major version.Minor version.Patchlevel"

Example

```
SAPGuiConnect(s_info, "testsap620");
SAPGuiVerCheckStr("6204.119.32");
```

SSL

SSL Commands

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 );
```

Return Value

Character: String containing the response from the secured server.

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 ) ;
```

Return Value

0 if the function is successful.
1 if the function is unsuccessful.

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 );
```

Return Value

Always returns 0

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 does not work. If you have an export grade license, only 40-bit codes work with your scripts. If you have a 128-bit license, all of the listed codes 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)
```

Return Value

1 if successful.
0 if unsuccessful.

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 uses 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);
```

Return Value

1 if successful.
0 if unsuccessful.

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" is used whenever the server requests one.

```
DO_SSLUseClientCert("qaload_cl");
```

DO_SSLUseClientCertPass

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);
```

Return Value

TRUE if successful.
FALSE if unsuccessful.

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 ) ;
```

Return Value

Always returns 0

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" );
...
...
```

TestPartner

TestPartner Commands

Since there are no convert or record options for a TestPartner session, these commands do not appear in the Function Wizard.

TP_Init

Used to create an instance of the Test Partner COM object. When an instance is created, the TestPartner application process is loaded and QALoad is able to interface with its functionality. This function must be called before any other function in this library.

TPExecuteScript

This function is used to execute a script within TestPartner.

TPExecuteVisualTest

This function is used to execute a visual test within TestPartner.

TPSetType

This function is used to set the value of a visual test parameter.

TP_Uninit
This function is called to release an instance of the TestPartner COM object. Once the instance is released, the TestPartner application process is closed. This function should be called when the test has been completed or aborted.

TP_Init

Used to create an instance of the TestPartner COM object. When an instance is created, the TestPartner application process is loaded and QALoad is able to interface with its functionality. This function must be called before any other function in this library.

TP_Init must occur exactly once per script and must be placed outside of the transaction loop.

Syntax

```
TP_Init(PLAYER_INFO *s_info);
```

Return Value

True/False

Parameters

Parameter	Description
s_info	A pointer to the PLAYER_INFO structure for this virtual user.

Example

```
TP_Init(s_info);
```

Note: Since there are no convert or record options for a TestPartner session, TestPartner commands do not appear in the Function Wizard.

TP_Uninit

This function is called to release an instance of the TestPartner COM object. Once the instance is released, the TestPartner application process is closed. This function should be called when the test has been completed or aborted.

TP_Uninit must occur exactly once per script, be placed outside of the transaction loop, and occur once in abort_function after printf.

Syntax

```
TP_Uninit();
```

Return Value

N/A

Parameters

None

Example

```
      END_TRANSACTION();

      TP_Uninit();
      EXIT();
}
int abort_function(PLAYER_INFO * s_info)
{
      printf("Task id: %i, Abort called!", S_task_id);

      TP_Uninit();
      EXIT();
}
```

Note: Since there are no convert or record options for a TestPartner session, TestPartner commands do not appear in the Function Wizard.

TPExecuteScript

This function is used to execute a script within TestPartner.

Syntax

```
TPExecuteScript(char * username, char * password, char * DSN, char * project, char *
script);
```

Return Value

True/False

Parameters

Parameter	Description
username	The user name to use when connecting to the TestPartner database.
password	The password to use when connecting to the TestPartner database.
DSN	The data from the DSNConfiguration.ini file in TestPartner 5.4 or 5.6. In TestPartner 5.1, 5.2, or 5.3, this data is from the ODBC System DSN (you might see a DSN name that does not belong to TestPartner). In TestPartner 6.0, this data is from the TestPartner COM object.
project	The project that exists in the associated DSN.
script	The script associated with the project.

Example

```
TPExecuteScript("Admin", "~encr~010E606B626660", "TestPartner", "Common" , "Script1");
```

Note: Since there are no convert or record options for a TestPartner session, TestPartner commands do not appear in the Function Wizard.

TPExecuteVisualTest

This function is used to execute a visual test within TestPartner.

Syntax

```
TPExecuteVisualTest(char * username, char * password, char * DSN, char * project, char * script);
```

Return Value

True/False

Parameters

Parameter	Description
username	The user name to use when connecting to the TestPartner database.
password	The password to use when connecting to the TestPartner database.
DSN	The data from the DSNConfiguration.ini file in TestPartner 5.4 or 5.6. In TestPartner 5.1, 5.2, or 5.3, this data is from the ODBC System DSN (you might see a DSN name that does not belong to TestPartner). In TestPartner 6.0, this data is from the TestPartner COM object.
project	The project that exists in the associated DSN.
script	The script (visual test) associated with the project.

Example

```
TPExecuteVisualTest("Admin", "admin", "TestPartner", "Common" , "Script1");
```

Note: Since there are no convert or record options for a TestPartner session, TestPartner commands do not appear in the Function Wizard.

TPSetType

This function is used to set the value of a visual test parameter.

Syntax

```
TPSetType(char * Type, char * ParamName, char * Value);
```

Return Value

True/False

Parameters

Parameter	Description
Type	Type of variable. Valid values are: Long, Double, Text, and Boolean

ParamName	Parameter data extracted from TestPartner.
Value	Value entered by user.

Example

```
TPSetType ("NumberLong", "numberVar1", "2653");
TPSetType ("NumberDouble", "numberVar2", "48.62");
TPSetType ("Text", "TextVar3", "Hello World");
TPSetType ("Boolean", "BoolVar1", "True");
```

Note: Since there are no convert or record options for a TestPartner session, TestPartner commands do not appear in the Function Wizard.

Winsock

Winsock Commands

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_IsWriteable
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.

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
Converts a string with escaped ^ control character sequences to raw text so that it can be manipulated.

AddrByte

Returns a byte of an internet address.

Only useful in very specific instances, particularly when scripting an FTP client that 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)
```

Return Value

Parameters

Parameter	Description
unsigned long address	The address of the client-side data port.
int which byte	The byte to send.

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.

Syntax

```
SOCKET DO_WSK_Accept(int nSocketHandle, int newSocketHandle)
```

Return Value

New socket handler if successful

1 if an error occurs

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);
```

Return Value

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);
```

Return Value

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

0 if successful
-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().

Syntax

```
int DO_WSK_Expect(int nConnectHandle, char *pattern)
```

Return Value

0 if the pattern was found

-1 if the timeout interval expired

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.

Syntax

```
int DO_WSK_ExpectAny(int nConnectHandle, int number_of_patterns, char *pattern1, ...)
```

Return Value

0-based index of the pattern that was matched first

-1 if not found.

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

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.

Syntax

```
int DO_WSK_ExpectAnyExpr(int nConnectHandle, int num_expressions, char *expression1, char *expression2, ...)
```

Return Value

Index of the pattern that was matched (0-based)

-1 if the timeout interval expired.

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

Waits for a unique pattern specified by a UNIX-style regular expression to occur. The pattern should signify the end of the response.

Syntax

```
int DO_WSK_ExpectExpr(int nConnectHandle, char *expression)
```

Return Value

0 if the pattern is found

-1 if the timeout interval expired

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)
```

Return Value

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;
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)
```

Return Value

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().

Syntax

```
int HexDecode(char *string)
```

Return Value

Number of bytes in the converted data (one half the number of input bytes)

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.

Syntax

```
int DO_WSK_Init(s_info)
```

Return Value

1 (one)

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,
WinsockIOCtlSocketCommandEnum command );
```

Return Value

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
argument	*WinsockIOCtlSocketCommandEnum* 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. Valid values are: Value — Description FIONBIO — Use with a nonzero second parameter to enable nonblocking mode FIONREAD — Use to determine the amount of data pending that can be read SIOCATMARK — Use to determine whether or not all OOB data has been 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 .

Syntax

```
int DO_WSK_IsReadable(int ConnectHandle);
```

Return Value

1 if the connection has data available to be read

0 if there is no data available

-1 if there was an error

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.

Syntax

```
int DO_WSK_IsWriteable(int ConnectHandle);
```

Return Value

1 if the connection is available for writing

0 if the output queue is full

-1 if there was an error

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.

Syntax

```
int DO_WSK_Listen(int nSocket);
```

Return Value

Always returns 0

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() reads 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.

Syntax

```
int DO_WSK_Quiet(int nConnectHandle, double seconds_of_quiet)
```

Return Value

Number of bytes read

-1 if an error was encountered

Parameters

Parameter	Description
nConnectHandle	The connection number.
seconds_of_quiet	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 used to build scripts that handle more complicated protocols.

Syntax

```
int DO_WSK_Read(int nConnectHandle, int bytes_to_read)
```

Return Value

Number of bytes read

-1 if an error is encountered

Parameters

Parameter	Description
nConnectHandle	The connection number.
bytes_to_read	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, WinsockRecvFlagsEnum flags, int * pnBytesRecv)
```

Return Value

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	*WinsockRecvFlagsEnum* Used to control the way in which the call is made. Valid values are: Value — Description MSG_PEEK — Peek flag MSG_OOB — OOB flag 0 — Normal
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, WinsockRecvFlagsEnum
flags, struct sockaddr *from_address, int * pnBytesRecv );
```

Return Value

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	*WinsockRecvFlagsEnum* Used to control the way in which the call is made. Valid values are: Value — Description MSG_PEEK — Peek flag MSG_OOB — OOB flag 0 — Normal
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.

Syntax

```
void DO_WSK_Reorder(int size, void *value)
```

Return Value

None

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);
```

Return Value

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);
```

```
free(Set1);
free(Set2);
free(Set3);
free(Time);
```

DO_WSK_Send

Sends data to a socket.

Syntax

```
DO_WSK_Send(int nConnectHandle, char *data)
```

Return Value

0 if successful

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, ...);
```

Return Value

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/jp eg, 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:\"82f2e16095cbc21:"
"973\"", "\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 );
```

Return Value

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,
WinsockSetSockOptionLevelEnum wsk_sockopt_optval );
```

Return Value

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	*WinsockSetSockOptionLevelEnum* Integer variable will receive the values of option_name upon function return. Valid values are: Value — Description SOL_SOCKET — Socket level IPPROTO_TCP — TCP level

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, WinsockShutdownMethodEnum shutdown_type );
```

Return Value

Parameters

Parameter	Description
nConnectHandle	A connection handle returned from a previous call to DO_WSK_Socket.
shutdown_type	*WinsockShutdownMethodEnum* Shutdown method options. Valid values are: Value — Description 0 — All receives are disabled

	1	All sends are disabled
	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, WinsockSocketTypeEnum type, int
protocol );
```

Return Value

Parameters

Parameter	Description
nConnectHandle	A connection handle to associate with a new socket.
address_family	The address family the socket uses. Refer to your Winsock documentation for a complete list.
type	*WinsockSocketTypeEnum* The Winsock socket type. Valid values are: Value — Description SOCK_DGRAM — UPD socket SOCK_STREAM — TCP socket
protocol	Specifies which protocol is 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.

Syntax

```
int DO_WSK_Write(int nConnectHandle, char *data_to_send, int bytes_to_write)
```

Return Value

Number of bytes written

-1 if an error was encountered

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.

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().

Syntax

```
char * EscapeStr(char *string, int count)
```

Return Value

A pointer to the converted characters

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 therefore, the script, uses DO_WSK_Bind() to bind a socket to an unspecified address 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.

Syntax

```
unsigned long GetLocalAddr(int nConnectHandle)
```

Return Value

Address of the local side of the connection

Parameters

Parameter	Description
nConnectHandle	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)
```

Return Value

Parameters

Parameter	Description
nConnectHandle	The connection number.

Example

```
unsigned short port;
...
port = GetLocalPort(S3);
```

GetRemoteAddr

Returns the port connected to on the remote side of a connection.

Syntax

```
unsigned long GetRemoteAddr(int nConnectHandle)
```

Return Value

Address of the remote side of the connection as a long integer

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)
```

Return Value

Parameters

Parameter	Description
nConnectHandle	The connection number.

Example

```
unsigned short port;
```

```
...
port = GetRemotePort(S3);
```

HiByte

Returns the high-order byte of the passed short integer.

Only useful in very specific instances, 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)
```

Return Value

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.

Only useful in very specific instances, 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)
```

Return Value

Parameters

Parameter	Description
port	The short value for which the low byte is returned.

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)
```

Return Value

Parameters

Parameter	Description
line_to_be_logged	A string to be written to the log file

Example

```
Log("Got here!");
```

MyByteOrder

Byte order of the machine running the script.

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).

Syntax

```
int MyByteOrder()
```

Return Value

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()
```

Return Value

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()
```

Return Value

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)
```

Return Value

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.

This 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.

Syntax

```
void ScanFloat(int byteorder, int length, char *buffer)
```

Return Value

None

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.

This 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.

Syntax

```
void ScanInt(int byteorder, int length, char *buffer)
```

Return Value

None

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);
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 is placed in the given pointer and count. This should be the address of an appropriate integral program variable, type-cast to a char *, and to be updated with the count.

Syntax

```
void ScanLenString(int byteorder, int length, char *count, char *buffer)
```

Return Value

None

Parameters

Parameter	Description
byteorder	The byteorder of the count value.
length	The size of the count value (1, 2, or 4).
Count	The address of a integral program value of the appropriate size to copy the count value to.

buffer	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.

Syntax

```
void ScanRewind( )
```

Return Value

None

Parameters

None.

Example

```
ScanRewind( );
```

ScanSkip

Skips the specified number of bytes in the scan buffer.

Syntax

```
void ScanSkip(int count)
```

Return Value

None

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.

Syntax

```
void ScanString(int length, char *buffer)
```

Return Value

None

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.

Syntax

```
int SetTimeout(int seconds)
```

Return Value

Previous timeout value

Parameters

Parameter	Description
seconds	The number of seconds to wait for a pattern in an DO_WSK_Expect or DO_WSK_ExpectAny, a connection in a DO_WSK_Accept, or a number of bytes to be received in a DO_WSK_Read.

Example

```
SetTimeout(20);
```

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)
```

Return Value

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 .

Syntax

```
int UnEscapeStr(char *string)
```

Return Value

Length of the converted 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 Commands

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_GetCitrixICAFile
Saves a WWW reply when its body is an ICA file.

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_GetCookiesForURL

Applies to HTTP and SSL requests. DO_GetCookiesForURL sends a message to the QALoad internal cookie storage requesting a list of cookies for this URL. The cookies are returned in a semicolon-separated list of cookies. Each cookie in the returned cookie list is put into the "name=value" form. The returned cookie list is suitable to be used as a cookie header for an HTTP request.

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_GetReplyBufferLength
Applies to HTTP and SSL requests. DO_GetReplyBuffer returns the length of 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_Https
Applies to SSL requests. Makes a secured request to the server specified by the http_statement.

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_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.

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_ProxyHttpVersion
Applies to HTTP and SSL requests. Specifies the version to use in proxy requests sent during playback. This affects whether or not the replies come back chunked. Only HTTP 1.1 requests receive chunked replies.

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_SetJavaScriptLevel
Applies to HTTP requests. Allows user to control the level of JavaScript execution for convert and replay.

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_SetPostDelay
Applies to HTTP requests. Sets how many seconds QALoad should wait for a reply from a server after the header has been sent for a POST request.

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.

DO_WWWFree
Applies to HTTP and SSL requests. Clears memory used by the script.

DO_WWWInitialize
Applies to HTTP and SSL requests. Sets all necessary internal variables needed to load test a WWW script.

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.

DownloadMediaWMP
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.

ExtractString
Applies to Visual Scripting. Retrieves data from the virtual browser. Will extract a text string from within the responses that came back from the last request. This command replaces the old method of extracting a string using the Get command.

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.

GetSiebelValue
Retrieves the value of a Siebel record from the Siebel library and saves it in a Local Variable.

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.

ModifyEncoding
ModifyEncoding is used in Visual scripts to convert strings to UTF8, EUCJP or to the language used by the script.

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.

SiebelInitialize
Initializes the Siebel correlation library and treats the current script file as a Siebel file. If the Siebel correlation library dll (ssdtcorr.dll) is not present in the QALoad directory, it throws an error in the TRACE window and exits the playback.

SiebelUpdatePage
Feeds the current response (response from last request) to the Siebel library. This tells the Siebel library to extract the Siebel parameter values from the response.

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.

CLoadString Index

CLoadString.Append1
Append the specified string to the current value of this CLoadString object.

CLoadString.Append2
Append the specified string to the current value of this CLoadString object.

CLoadString..Append3
Append the specified string to the current value of this CLoadString object.

CLoadString.AsInteger
Returns an integer representation of the string value.

CLoadString.Assign1
Set s the value of the CLoadString object.

CLoadString.Assign2
Sets the value of the CLoadString object.

CLoadString.Assign3
Sets the value of the CLoadString object.

CLoadString.Assign4
Sets the value of the CLoadString object.

CLoadString.AsString
Returns a representation of the CLoadString value as a string.

CLoadString.Clear
Replaces the value of the CLoadString object with an empty string.

CLoadString.CLoadString1
Constructs a new CLoadString object. An initial value can be specified as an argument.

CLoadString.CLoadString2
Constructs a new CLoadString object. An initial value can be specified as an argument.

CLoadString.CLoadString4
Constructs a new CLoadString object. An initial value can be specified as an argument.

CLoadString.CLoadString5
Constructs a new CLoadString object. An initial value can be specified as an argument.

CLoadString.Compare1
Compares two strings. This method returns 0 if the two strings are equal.

CLoadString.Compare2
Compares two strings. This method returns 0 if the two strings are equal.

CLoadString.GetLength
Returns the length of the string.

CLoadString.IsEmpty
Returns true if the string value is empty, false otherwise.

CLoadString.Lowercase
Converts the CLoadString object's value to lowercase.

CLoadString.Mid
Returns a CLoadString with the value of the substring corresponding to the specified index and length parameters.

CLoadString.SetSeparator
Sets the CLoadString object's separator string for use in conjunction with the Token() method.

CLoadString.Token
Returns a pointer to a CLoadString object containing the token specified by the index and the separator string (which is set by using the SetSeparator method). The default separator, if none is specified, is the pipe character ('|').

CLoadString.Constructor
Constructs a new CLoadString object. An initial value can be specified as an argument.

CLoadString.Operator!=
Returns true if the two string values are not equal.

CLoadString.Operator[]
Allows access to the characters in the string.

CLoadString.Operator+
Returns a CLoadString that has the value of the two strings concatenated together.

CLoadString.Operator+=
Returns a reference to a CLoadString that has the value of the two strings concatenated together.

CLoadString.Operator=
Sets the string value of the CLoadString object.

CLoadString.Operator==
Returns true if the left and right values are equal.

Attach

Applies to Visual Scripting. Supported in HTML mode only. 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 ( const page_id& page );
```

Return Value

True if the page was attached to.
False if not.

Parameters

Parameter	Description
page	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 ( WWWClearEnum type );
```

Return Value

True if cleared successfully.
False if not cleared successfully .

Parameters

Parameter	Description
type	*WWWClearEnum*

The type of clear to do. Valid types are listed in the following tables:

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)
INTERNAL_BUFFERS	Clear any internal buffers.
JAVASCRIPT_ENGINE	Clear the Javascript state.
NTLM_AUTHORIZATION	Clear the Integrated Windows Authentication (formerly 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.

Versions

Versions of Click_On are:

boolean Click_On (WWWClickOnLinkEnum link, string description);

boolean Click_On (WWWClickOnLinkEnum link, integer count);

boolean Click_On (WWWClickOnLinkEnum link, WWWClickOnSpecifierEnum specifier, string description);

boolean Click_On (WWWClickOnLinkEnum link, integer count, WWWClickOnSpecifierEnum specifier, string description);

boolean Click_On (WWWClickOnLinkEnum link, integer count, WWWClickOnSpecifierEnum specifier, string description, string x_coord, string y_coord);

boolean Click_On (WWWClickOnLinkEnum link, integer count, WWWClickOnSpecifierEnum specifier, string description, WWWClickOnOptionEnum click_option);

boolean Click_On (WWWClickOnLinkEnum link, integer count, WWWClickOnSpecifierEnum specifier, string description, string region);

About CLoad String

CLoadString is a C++ class that represents a string. It has a number of methods that allow access to and modification of the string value.

Declaration

CLoadString variables can be declared by specifying CLoadString as the type and a variable name. Initial values for the variable can be specified in several ways:

```
char *fish = "trout";
CLoadString a(fish);    // Initial value is "trout"
CLoadString eleven(11); // Initial value is "11"
CLoadString b = "blue"; // Initial value is "blue"
CLoadString string1;    // No initial value, length is zero
```

Modification of the string value

The string value held by the CLoadString variable can be modified by using its methods or operators. For example, string values can be appended to a CLoadString object by either using the addition operator, the addition assignment operator or the Append() method:

```
CLoadString s = "abc";
CLoadString two = s + "123"; // Addition operator produces "abc123"
s += "def"; // Use the addition assignment operator
s.Append("g"); // or use the Append() method.
// After this statement, the value is "abcdefg"
```

Comparison

There are few ways to perform comparison on CLoadString objects. The Compare() method will return an int value in much the same manner as the C function strcmp():

```
CLoadString s = "abcdefg";
if (s.Compare("abcdefg") == 0)
{
    // string value was "abcdefg"
}
if (s.Compare("alphabet") != 0)
{
    // string value was not equal to "alphabet"
}
```

Or comparison can be done by using the equal and not equal operators:

```
CLoadString s = "abcdefg";
if (s == "abcdefg")
{
    // string value was "abcdefg"
}
if (s != "alphabet")
{
    // string value was not equal to "alphabet"
}
```

Interaction between CLoadString and char types

The CLoadString class is easily interchangeable with string literals (const char *). For example:

```
CLoadString cat("cat");
RR__printf("the value of cat is '%s'\n", cat);
OUTPUT: the value of cat is 'cat'
```

CLoadString.CLoadString

Constructs a new CLoadString object. An initial value can be specified as an argument.

Syntax

```
CLoadString
```

Example

```
CLoadString blank; // the value of blank is ""
```

CLoadString.CLoadString

Constructs a new CLoadString object. An initial value can be specified as an argument.

Syntax

```
CLoadString(const CLoadString& strValue);
```

Parameters

Parameter	Description
strValue	An initial value for the new CLoadString object.

Example

```
CLoadString a("a"); // the value of a is "a"
CLoadString aa(a); // the value of aa is "a"
CLoadString airplane("airplane", 3);// the value of airplane is "air"
CLoadString number(31236); // the value of number is "31236"
```

CLoadString.CLoadString

Constructs a new CLoadString object. An initial value can be specified as an argument.

Syntax

```
CLoadString(const char*, int nLength);
```

Parameters

Parameter	Description
sValue	An initial value for the new CLoadString object
nLength	Length of initial value

Example

```
CLoadString airplane("airplane", 3);// the value of airplane is "air"
```

CLoadString.CLoadString

Constructs a new CLoadString object. An initial value can be specified as an argument.

Syntax

```
CLoadString(int nValue);
```

Parameters

Parameter	Description
nValue	A numeric initial value for this CLoadString object. The number will be converted into a string internally.

Example

```
CLoadString number(31236);// the value of number is "31236"
```

CLoadString.Append

Append the specified string to the current value of this CLoadString object.

Syntax

```
const CLoadString& Append (const CLoadString&);
```

Return Value

CLoadString& - result of append operation

Parameters

Parameter	Description
strValue	String to append to the value of this CLoadString

Example

```
CLoadString abc("abc"); // Initial value is "abc"
CLoadString def("def");
```

```
abc.Append(def);// Value is now "abcdef"
```

CLoadString.Append

Append the specified string to the current value of this CLoadString object.

Syntax

```
const CLoadString& Append (const char*sValue);
```

Return Value

CLoadString& - result of append operation

Parameters

Parameter	Description
sValue	String to append to the value of this CLoadString

Example

The following is an example of appending a value to a CLoadString object

```
CLoadString abc("abc"); // Initial value is "abc"
CLoadString def("def");

abc.Append(def);// Value is now "abcdef"

def.Append("g");// Value is now "defg"
```

CLoadString.Append

Append the specified string to the current value of this CLoadString object.

Syntax

```
const CLoadString& Append (const char*sValue, int nLength);
```

Return Value

CLoadString& - result of append operation

Parameters

Parameter	Description
sValue	String to append to the value of this CLoadString
nLength	Length of the string value to append

Example

```
CLoadString abc("abc"); // Initial value is "abc"
CLoadString def("def");

abc.Append(def);// Value is now "abcdef"

def.Append("g");// Value is now "defg"
```

```
abc.Append("ghijklmnop", 1);// Value is now "abcdefg"
```

CLoadString.AsInteger

Returns an integer representation of the string value.

Syntax

```
int AsInteger() const;
```

Return Value

int – Integer representation of CLoadString value

Parameters

Example

```
CLoadString eleven(11);// Value is "11"
CLoadString fifteen("15");
int iEleven = eleven.AsInteger();// iEleven == 11
int iFifteen = fifteen.AsInteger();// iFifteen == 15
```

CLoadString.Assign

Set s the value of the CLoadString object

Syntax

```
const CLoadString& Assign (const CLoadString& strValue);
```

Return Value

const CLoadString & - CLoadString object with the specified value

Parameters

Parameter	Description
strValue	Value to assign to the CLoadString object

Example

```
CLoadString str;
CLoadString b("blue");

str.Assign(b);// The value of str is "blue"
```

CLoadString.Assign

Sets the value of the CLoadString object

Syntax

```
const CLoadString& Assign (const char*sValue, int nLength);
```

Return Value

const CLoadString& - CLoadString object containing the specified value

Parameters

Parameter	Description
sValue	Value to assign to the CLoadString object
nLength	The length of the value parameter

Example

```
CLoadString str;
CLoadString b("blue");

str.Assign("greenish", 5);// The value of str is "green"
```

CLoadString.Assign

Sets the value of the CLoadString object

Syntax

```
const CLoadString& Assign (const char*);
```

Return Value

const CLoadString& - CLoadString containing the specified value

Parameters

Parameter	Description
sValue	Value to assign to the CLoadString object

Example

```
CLoadString str;

str.Assign("red");
```

CLoadString.Assign

Sets the value of the CLoadString object

Syntax

```
const CLoadString& Assign (int nValue);
```

Return Value

const CLoadString & - CLoadString containing the value specified

Parameters

Parameter	Description

nValue	Numeric value for the CLoadString object

Example

```
CLoadString str;

str.Assign(1234);// The value of str is "1234"
```

CLoadString.AsString

Returns a representation of the CLoadString value as a string.

Syntax

```
const char* AsString() const;
```

Return Value

const char * - CLoadString object with the value specified

Example

```
const char *b3 = "b3";
CLoadString thirtyone(31);
CLoadString bthree(b3);

if (0 == strcmp(thirtyone.AsString(), "31"))
{
}
if (thirtyone == "31")
{
}
if (0 == strcmp(bthree.AsString(), b3))
{
}
if (bthree == b3)
{
}
```

CLoadString.Clear

Replaces the value of the CLoadString object with an empty string.

Syntax

```
voidClear ();
```

Example

```
CLoadString empty("not empty");
empty.Clear();// The value of empty is now ""
```

CLoadString.Compare

Compares two strings. This method returns 0 if the two strings are equal.

Syntax

```
int Compare (const char* sValue)
```

Parameters

Parameter	Description
sValue	String to compare this CLoadString object's value to.

Example

```
CLoadString dog = "dog";
CLoadString cat("cat");

if (dog.Compare(cat) != 0)
{
// this is true
}
if (cat.Compare("cat") == 0)
{
// this is true
}
```

CLoadString.Compare

Compares two strings. This method returns 0 if the two strings are equal.

Syntax

```
int Compare (const CLoadString& sValue)
```

Parameters

Parameter	Description
sValue	String to compare this CLoadString object's value to.

Example

```
CLoadString dog = "dog";
CLoadString cat("cat");

if (dog.Compare(cat) != 0)
{
// this is true
}
if (cat.Compare("cat") == 0)
{
// this is true
}
```

CloadString.Constructor

Constructs a new CLoadString object. An initial value can be specified as an argument.

Syntax

```
CLoadString ();

CLoadString (const CLoadString&);
```

```
CLoadString (const char*);
CLoadString (const char*, int nLength);
CLoadString (int nValue);
```

Example

```
CLoadString a("a"); // the value of a is "a"
CLoadString aa(a); // the value of aa is "a"
CLoadString airplane("airplane", 3);// the value of airplane is "air"
CLoadString number(31236); // the value of number is "31236"
```

CLoadString.GetLength

Returns the length of the string.

Syntax

```
int GetLength () const;
```

Return Value

int – The length of the string.

Parameters

Example

```
CLoadString empty("empty");
empty.Clear();// The value of empty is now ""
if (empty.GetLength() == 0)
{
}
```

CLoadString.IsEmpty

Returns true if the string value is empty, false otherwise.

Syntax

```
bool IsEmpty () const;
```

Return Value

bool – true if the string value is empty, false otherwise

Parameters

Example

```
CLoadString empty("");
if (empty.IsEmpty() == true)
{
}
```

CLoadString.Lowercase

Converts the CLoadString object's value to lowercase.

Syntax

```
voidLowercase ();
```

Example

```
CLoadString str1("SoMeThInG");
CLoadString str2("lower?");
CLoadString str3("Title");
str1.Lowercase();// value is now "something"
str2.Lowercase();// value is now "lower?"
str3.Lowercase();// value is now "title"
```

CLoadString.Mid

Returns a CLoadString with the value of the substring corresponding to the specified index and length parameters.

Syntax

```
CLoadString Mid ( int nIndex, int nLength= -1 ) const;
```

Return Value

CLoadString - CLoadString with the value of the substring corresponding to the specified index and length parameters.

Parameters

Parameter	Description
nIndex	int nIndex – Zero based index for the beginning of the substring
nLength	int nLength – If specified, this is the number of characters that the substring should contain. If it is not specified, the substring will contain all characters from the index to the end of the string.

Example

```
const char *txt = "With QALoad, you can simulate load...";
CLoadString text = txt;
CLoadString qaload = text.Mid(5, 6); // the value of qaload is "QALoad"
CLoadString blurb = text.Mid(13); // value is "you can simulate load..."
```

CLoadString.Operator[]

Allows access to the characters in the string.

Syntax

```
char operator [] ( int ) const;
```

Return Value

char – the character at the specified index

Parameters

int – Zero based index into the string value

Example

```
CLoadString apple("apple"); // apple[0]=='a', apple[4]=='e'
if (apple[0] == 'a')
```

```
{
}
```

CLoadString.Operator+

Returns a CLoadString that has the value of the two strings concatenated together.

Syntax

```
CLoadString operator + ( const CLoadString& ) const;
CLoadString operator + ( const char* ) const; // Combine the strings
```

Return Value

Returns a CLoadString that has the value of the two strings concatenated together.

Example

```
CLoadString crab("crab");
CLoadString apple("apple");
const char *tree = "tree";
CLoadString crabapple = crab + apple; // value of crabapple is "crabapple"
CLoadString crabtree = crab + tree;   // value of crabtree is "crabtree"
```

CLoadString.Operator+=

Returns a reference to a CLoadString that has the value of the two strings concatenated together.

Syntax:

```
const CLoadString& operator += ( const CLoadString& );
const CLoadString& operator += ( const char* );
```

Return Value

Returns a reference to a CLoadString that has the value of the two strings concatenated together.

CLoadString.Operator=

Sets the string value of the CLoadString object.

Syntax

```
const CLoadString& operator = ( const CLoadString& );
const CLoadString& operator = ( const char* );
```

Returns

Returns a reference to a CLoadString that has the value of the two strings concatenated together.

Example

```
CLoadString str = "str";
```

CLoadString.SetSeparator

Sets the CLoadString object's separator string for use in conjunction with the Token() method.

The default separator, if none is specified, is the pipe character ('|').

Syntax

```
void SetSeparator ( const char* szSeparator );
```

Parameters

Parameter	Description
szSeparator	const char* szSeparator – Specifies the string between tokens

Example

```
CLoadString data = "MIDDLEWARE|WWW|5.8.0.140|Full Encryption";
data.SetSeparator("|WWW|");

CLoadString *sMW = data.Token(0); // sMW is "MIDDLEWARE"
RR__printf("sMw='%s'\n", sMW->AsString());
delete sMW;
```

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 *QA*Load Player main window, in the Run As: group, select the Process option.
- For streaming media playback, *QA*Load requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing *QA*Load " chapter of the *QA*Load Installation and Configuration Guide.

Syntax

```
DisableStatisticsRP();
```

Return Value

Parameters

None.

Example

```
DisableStatisticsRP();
```

CLoadString.Operator==

Returns true if the left and right values are equal.

Syntax

```
bool operator == ( const CLoadString& ) const;

bool operator == ( const char* ) const;
```

Return Value

bool - Returns true if the two string values are the same.

Parameters

Example

```
CLoadString b("blue");
if (str == "blue")
{
    // this is true
}
```

CLoadString.Operator!=

Returns true if the two string values are not equal.

Syntax

```
bool operator != ( const CLoadString& ) const;
bool operator != ( const char* ) const;  // Are the strings not equal
```

Return Value

bool - Returns true if the two string values are not equal

Example

```
CLoadString b("blue");
if (str != "green")
{
    // this is true
}
```

CLoadString.Token

Returns a pointer to a CLoadString object containing the token specified by the index and the separator string (which is set by using the SetSeparator method). The default separator, if none is specified, is the pipe character ('|').

The pointer value returned by the Token method must be manually deleted by the caller by using the delete operator

Syntax

```
CLoadString* Token (int nCount= -1);
```

Return Value

CLoadString* - A pointer to a CLoadString object containing the token specified by the index and the separator string

Parameters

Parameter	Description

nCount	int nCount – If specified, zero based index for the token requested. If not specified or -1, the method returns the next token.

Example

```
CLoadString data = "MIDDLEWARE|WWW|5.8.0.140|Full Encryption";
data.SetSeparator("|WWW|");
CLoadString *sMW = data.Token(0);      // sMW is "MIDDLEWARE"
RR__printf("sMw='%s'\n", sMW->AsString());
delete sMW;
// Iterate and print all of the tokens
CLoadString data2 = "MIDDLEWARE|Citrix|5.8.0.140|Full Encryption";
data2.SetSeparator("|");
CLoadString *x = NULL;
for(;;)
{
      x = data2.Token();
      if (NULL == x)
      {
            break;
      }
      RR__printf("x='%s'\n", x->AsString());
      delete x;
}
```

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);
```

Return 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_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_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 )
```

Return Value

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=\"phylename\"; 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);
```

Return Value

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 );
```

Return Value

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)

Return Value

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);
```

Return Value

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 ( WWWClearEnum type );
```

Return Value

Parameters

Parameter	Description
type	*WWWClearEnum* The type of clear to do. Valid types are listed in the following tables: Values for Meta type 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)
INTERNAL_BUFFERS	Clear any internal buffers.
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

```
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();
```

Return Value

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()
```

Return Value

Parameters

None.

Example

```
...
...
/* Request: 1 */
DO_Http ("GET http://company.com/ HTTP/1.0\r\\n\r\n" );
DO_ClearDNSCache();
/* Request: 2 */
/*
* 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();

Return Value

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/dynredir.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/dynredir.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/ dynredir.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.

Note: The DO_EnableJavascript command must appear after the DO_AutomaticSubRequests command in the 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_AutomaticSubRequests(TRUE);
...
...
DO_EnableJavascript(TRUE);
...
...
```

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.

Caution: The string buffer parameter variable should be explicitly initialized to NULL. Failure to do so results in a memory error in the script.
Once the string buffer has been allocated, it can be reused within the same transaction loop without being explicitly freed. However, the buffer memory should be freed at the end of the transaction loop. Failure to free the memory buffer at the end of the transaction loop results in a memory leak.

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. 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( );
```

Return Value

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. Fox 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.

This functionality is available in HTML mode and HTTP modes with parsing turned on.

Note: If you are adding commands, QALoad uses the following rules for matching the anchorName parameter to the <img> 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 <img> tag in the source HTML, use the alt= attribute.

  Example:
  FOR: click<img alt="look" src="look.gif">
  USE: DO_GetAnchorHREF ("alt=look", Anchor [0]) ;
- If the <img> tag has no alt= attribute, use the src= attribute.

  Example:
  FOR: click<img src="look.gif">
  USE: DO_GetAnchorHREF ("src=look.gif", Anchor [0]) ;
- If there is no <img> 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 <img> 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"
        "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 <img> 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_GetCitrixICAFile

Saves a WWW reply when its body is an ICA file.

This function will save a WWW reply when its body is an ICA file. If the reply is an ICA file then the ICA file will be saved to the "QALoad\ BinaryFiles" directory with the following naming convention: "script name_vuNN_XX.ica", where NN is the "absolute virtual user number" and NN is the transaction number. This file location can then be used within Citrix CtxConnectICA() command.

Syntax

```
int DO_GetCitrixICAFile (char ** szFileName);
```

Return Value

Returns 1 if the function call was successful, else 0 if an error occurs.

Parameters

Parameter	Description
szFileName	Address to a string that specifies the location of the ICA that was sent back as part of the server reply.

Example

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

```
…
…
BEGIN_TRANSACTION();
…
…
DO_Http("GET http://citrixserver/ica/notepad/ HTTP/1.0\r\n\r\n")
DO_GetCitrixICAFile(&strICAFileName[0]);
…
…
RR__printf("ICA file = %s",strICAFileName[0]);
…
CtxConnectICA(strICAFileName[0];
…
…
for(i=0; i<1; i++)
{
if(strICAFileName[i] != NULL)
unlink(strICAFileName[i]);
free(strICAFileName[i]);
strICAFileName[i]=NULL;
}
…
…
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_DynanmicCookieHandling be set to TRUE.

Syntax

BOOL DO_GetCookie (const char * szName, int nCount, char ** pszCookie

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 * aspsessionid;
...
...
BEGIN_TRANSACTION();
...
...

/* Request: 1 */

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

/*
* Get a cookie named USER_ID
*/
DO_GetCookie ( "USER_ID", 1, &userid );

/*
* Get the second ASPSESSIONID cookie cookie. ASPSESSIONID
* cookies always have extra characters on the end to make
* them unique.
*
* An example ASPSESSIONID: ASPSESSIONIDQQQGGQDO=EBOOONBBFH
* BBELAJIMEFAKAP
*/
DO_GetCookie ("ASPSESSIONID*", 2, &aspsessionid );
```

DO_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_GetCookiesForURL

Applies to HTTP and SSL requests. DO_GetCookiesForURL sends a message to the QALoad internal cookie storage requesting a list of cookies for this URL. The cookies are returned in a semicolon-separated list of cookies. Each cookie in the returned cookie list is put into the "name=value" form. The returned cookie list is suitable to be used as a cookie header for an HTTP request.

Note: DO_GetCookiesForURL requires DO_DynanmicCookieHandling be set to TRUE.

Syntax

```
BOOL DO_GetCookiesForURL ( const char * szURL, char ** pszCookie );
```

Return Value

TRUE for successful

FALSE for unsuccessful

Parameters

Parameter	Description
szURL	The requested URL
pszCookies	Address of a string pointer to hold the cookies.

Example

```
char * cookielist= NULL;
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
/*
 * In this request two cookies will be set. CookieA will have the value of ValueA and
 * CookieB will have the value of ValueB.
 *
 * Set-Cookie: CookieA=ValueA; domain=.company.com; path=/;
 * Set-Cookie: CookieB=ValueB; domain=.company.com; path=/;
 */
DO_Http ( "GET http://www.company1.com/ HTTP/1.0\r\n\r\n" );
/*
 * Get all cookies set for http://www.company1.com/. After this call cookielist will have
 * the value "CookieA=ValueA; CookieB=ValueB.
 */
DO_GetCookiesForURL ( "http://www.company1.com/", &cookielist );
/*
 * Now make a request to company 2 with all the cookies from www.company1.com to
 * www.company2.com.
 */
DO_SetValue ( "CompanyCookies", cookielist );
DO_Http ( "GET http://www.company2.com/ HTTP/1.0\r\n"
        "Cookie: {*CompanyCookies}\r\n\r\n" );
```

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.

This functionality is available in HTML mode and HTTP modes with parsing turned on.

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\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.

Subsequently, this value can be used 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.

This functionality is available in HTML mode and HTTP modes with parsing turned on.

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_GetReplyBufferLength

Applies to HTTP and SSL requests. DO_GetReplyBuffer returns the length of the HTTP response from the last DO_Http request.

Syntax

```
int DO_GetReplyBufferLength()
```

Return Value

The length of the last HTTP reply or 0 if there is no response found.

Parameters

None.

Example

```
int bufLength;
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http("GET http://company.com/ HTTP/1.0\r\n\r\n" );
bufLength = DO_GetReplyBufferLength();
...
...
```

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.

DO_GetUniqueString allocates enough space in the parameter passed in as the string buffer to hold the string (including the NULL). Please remember to free any memory after using the returned string. Any memory created with this command that is not explicitly freed results in a memory leak.

Note: If the string buffer parameter has a non-NULL value when passed to this function, the memory is leaked.

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( );
```

Return Value

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_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 );
```

Return Value

Character: String containing the response from the secured server.

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_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(WWWHTTPVersionEnum version);
```

Parameters

Parameter	Description
version	*WWWHTTPVersionEnum* The HTTP version. If specified as Auto, the version used for each request is determined from the request. Valid values are: Value — Description "1.0" — HTTP version 1.0 "1.1" — HTTP version 1.1 "AUTO" — HTTP version is set in DO_Http and DO_Https

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);
```

Return Value

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 Integrated Windows Authentication.

DO_NTLMAuthorization is related to the Integrated Windows Authentication (formerly NTLM) option on the QALoad Script Development Workbench Record Advanced Options. 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 an Integrated Windows Authentication controlled site, it uses the Integrated Windows Authentication user ID and password that are provided to access that site.

Note: Integrated Windows Authentication user names and passwords can be variablized by machine, but not by user.

Syntax

```
DO_NTLMAuthorization(const char *name, const char *password);
```

Return Value

Parameters

Parameter	Description
name	A valid user ID for the Integrated Windows Authentication-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_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);
```

Return Value

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, 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_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_ProxyHttpVersion()

Applies to HTTP and SSL requests. Specifies the version to use in proxy requests sent during playback. This affects whether or not the replies come back chunked. Only HTTP 1.1 requests receive chunked replies.

Do_ProxyHttpVersion is related to the Proxy HTTP Version option on the General page of the WWW Convert Options dialog box. The default setting is 1.0.

Syntax

```
DO_ProxyHttpVersion (const char *version)
```

Return Value

Parameters

Parameter	Description
version	The HTTP version ("1.0" or "1.1").

Example

```
DO_ProxyHttpVersion("1.0");
```

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);
```

Return Value

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);
```

Return Value

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

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_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)
```

Return Value

Parameters

Parameter	Description
nThreshold	Number of JavaScript evaluations to make before cleaning up the JavaScript engine.

Example

```
...
...
DO_SetJavascriptCleanupThreshold(200);
...
...
```

DO_SetJavaScriptLevel

Applies to HTTP requests. Allows user to control the level of JavaScript execution for convert and replay.

Syntax

```
DO_SetJavaScriptLevel (WWWJavascriptExecutionLevelEnum level );
```

Return Value

Parameters

Parameter	Description
level	*WWWSetJavaScriptLevelEnum* JavaScript execution level user sets in the WWW Advanced convert dialog. Valid values are: Value — Description FULL — Create/execute scripts that parse through and handle JavaScript. LIMITED — Create/execute scripts that handle the JavaScript to the same level as QALoad 5.2. NONE — Create/execute scripts that disregard all JavaScript.

Examples

```
DO_SetJavaScriptLevel ( FULL );
DO_SetJavaScriptLevel ( LIMITED );
DO_SetJavaScriptLevel ( NONE );
```

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);
```

Return Value

Parameters

Parameter	Description
count	The number of connections to make. *QA*Load accepts 1-4. 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)
```

Return Value

Parameters

Parameter	Description
nValue	The default is 4.

Example

```
...
...
BEGIN_TRANSACTION();
DO_SetMaximumRetries(5);
...
...
```

DO_SetPostDelay

Applies to HTTP requests. Sets how many seconds QALoad should wait for a reply from a server after the header has been sent for a POST request.

DO_SetPostDelay sets will send the header and body of a POST request all at once if set to zero, or it will wait up to the specified number of seconds for the server to respond before sending the body.

Syntax

```
void DO_SetPostDelay( long delay );
```

Return Value

None

Parameters

Parameter	Description
delay	Number of seconds to delay between sending the header and body of a POST request.

Example

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)
```

Return Value

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 fails 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 range of values is 5 to 65535. The default is 120 seconds.

Syntax

```
DO_SetTimeout(int timeout);
```

Return Value

Parameters

Parameter	Description
timeout	The number of seconds to wait. Range of values is 5 to 65535. 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 );
```

Return Value

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 ) ;
```

Return Value

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 always terminates the current persistent connection if one is present. This allows 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.

Note: When called with a proxy server that is not found on the network, DO_UseProxy aborts even when you select "Continue executing and ignore the error" in the Error Handling Options dialog box. See Anticipating Error Conditions for more information on setting error handling options.

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 uses the PAC script to determine which proxy, if any, to connect to hosts.

Syntax

```
BOOL DO_UseProxyAutomaticConfiguration ( const char * szUrl );
```

Return Value

TRUE = successful
FALSE = unsuccessful

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, WWWVerifyDocTitleComparisonTypeEnum nType ) ;
```

Return Value

Integer value
1 = match found. Indicated by the Player debug window.
0 =match not found. The function calls WWW_FATAL_ERROR, which either aborts the test or continues, 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 *QA*Load Script Development Workbench Convert Options wizard.
nType	*WWWVerifyDocTitleComparisonTypeEnum* Type corresponding to the comparison options available on the QALoad Script Development Workbench Convert Options wizard. Valid values are: Value — Description TITLE — Verify title PREFIX — Verify title prefix SUFFIX — Verify title suffix

Example

```
DO_Http ( http_statement ) ;
DO_VerifyDocTitle ( "Welcome to Compuware" , TITLE ) ;
```

DO_WWWFree

Applies to HTTP and SSL requests. Clears memory used by the script.

This command is used at the end of every HTTP script. DO_WWWFree is automatically inserted during the convert process and should never need to be adjusted.

Syntax

```
DO_WWWFree( );
```

Return Value

Parameters

None.

Example

```
...
...
END_TRANSACTION();
DO_WWWFree();
REPORT(SUCCESS);
```

DO_WWWInitialize

Applies to HTTP and SSL requests. Sets all necessary internal variables needed to load test a WWW script.

Use this command at the beginning of every WWW script, but never more than once in a script.

Note: This function should be written exactly as shown below.

Syntax

```
DO_WWWInitialize(PLAYER_INFO *sInfo, WWWConvertModeEnum convertMode);
```

Return Value

Parameters

Parameters	Description
sInfo	A pointer to a PLAYER_INFO memory structure.
convertMode	*WWWConvertModeEnum* Convert mode options. Valid values include: Value — Description HTML_MODE — The script will be run in HTML mode. HTTP_MODE — The script will be run in HTTP mode.

Example

```
...
...
int rhobot_script(s_info)
PLAYER_INFO *s_info;
{
...
...
DO_WWWInitialize(s_info, HTML_MODE);
...
...
BEGIN_TRANSACTION();
...
...
}
```

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 QALoad Installation and Configuration Guide.

Syntax

```
DownloadMediaFromASX( int secDuration );
```

Return Value

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 *QA*Load Script Development Workbench.
- Real Networks streaming media is only supported in process mode. On the *QA*Load Player main window, in the **Run As:** group, select the **Process** option.
- For streaming media playback, *QA*Load requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing *QA*Load" chapter of the *QALoad Installation and Configuration Guide*.

Syntax

```
DownloadMediaRP( char *URL, int timeout );
```

Return Value

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 may be longer than the timeout.

Example

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

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 QALoad Installation and Configuration Guide.

Syntax

```
DownloadMediaWMP( char *reqURL, int secDuration );
```

Return Value

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 *QA*Load Player main window, in the Run As: group, select the Process option.
- For streaming media playback, *QA*Load requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing *QA*Load " chapter of the *QALoad Installation and Configuration Guide*.

Syntax

```
EnableStatisticsRP( WWWRPStatsMaskEnum flags, int interval, BOOL traceOutput );
```

Return Value

Parameters

Parameter	Description
`flags`	*WWWRPStatsMaskEnum* The flag values in the following table can be combined using a logical OR. Flag values include: Values for Meta flags

Type	Description
`QAL_WWW_RN_STAT_ALL_LEVELS`	`All statistic levels.`
`QAL_WWW_RN_STAT_PLAYER`	`Player level statistics.`
`QAL_WWW_RN_STAT_SOURCE`	`Source level statistics.`
`QAL_WWW_RN_STAT_STREAM`	`Stream level statistics.`
`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.
interval	Report every *nth* stat received.
traceOutput	TRUE means send enabled stats to *QA*LOad Player window (if *QA*LOad 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 );
```

ExtractString

Applies to Visual Scripting. Retrieves data from the virtual browser.

Will extract a text string from within the responses that came back from the last request. This command replaces the old method of extracting a string using the Get command.

Syntax

string ExtractString (int *nOccurrence*, string *strPrecede*, string *strFollow*, int *nSearchType*);

Return Value

Returns the extracted string. If the extracted string is not found then an appropriate exception will be thrown and caught which will produce an error message specifying why the string was not found.

Parameters

Parameter	Description
nOccurrence	WWWExtractStringNEnum

<table>
<tr><td></td><td>This field specifies which occurrence of the search strings to find before extracting the text between them. Below are the various string formats you can use to specify the occurrence.
<table><tr><th>Type</th><th>Description</th></tr><tr><td>N</td><td>This will find the Nth occurrence of the search strings within the responses.</td></tr><tr><td>EXTRACT_LAST</td><td>This will find the Last occurrence of the search strings within the responses.</td></tr><tr><td>EXTRACT_LAST - N</td><td>This will find the (Last-Nth) occurrence of the search strings within the responses.</td></tr><tr><td>EXTRACT_RANDOM</td><td>This will calculate a random number between 1 and Last and find this occurrence of the search strings within the responses.</td></tr></table></td></tr>
<tr><td>strPrecede</td><td>The Left search string which will precede the text which is to be extracted.</td></tr>
<tr><td>strFollow</td><td>The Right search string which will follow the text which is to be extracted.</td></tr>
<tr><td>nSearchType</td><td>WWWExtractStringSearchtypeEnum
Valid values are:
<table><tr><th>Specifier</th><th>Description</th></tr><tr><td>INCLUDE_REDIRS_AND_ADDSUBS</td><td>Search includes not only the main page, its frames and subrequests but also Redirects and Additional Subrequests. This is the only search method for HTTP scripts.</td></tr><tr><td>NO_REDIRS_AND_ADDSUBS</td><td>Search only includes the main page, its frames, and its subrequests. This is the default value for HTML scripts.</td></tr></table></td></tr>
</table>

Example

The following are examples of using the ExtractString command

```
CLoadString myString = "";
myString = ExtractString (1, "SessionID=\"", "\"", INCLUDE_REDIRS_AND_ADDSUBS);
myString = ExtractString (3, "abc", "def", NO_REDIRS_AND_ADDSUBS);
myString = ExtractString (EXTRACT_LAST, "abc", "def", NO_REDIRS_AND_ADDSUBS);
myString = ExtractString (EXTRACT_LAST - 2, "abc", "def", NO_REDIRS_AND_ADDSUBS);
myString = ExtractString (EXTRACT_RANDOM, "abc", "def", NO_REDIRS_AND_ADDSUBS);
```

Fill_In

Applies to Visual Scripting. Used to represent how the user filled in fields on a form before clicking on a submit button.

Versions

Versions for Fill_In are:

boolean Fill_In (WWWControlEnum control_type, string description, string value);

boolean Fill_In (WWWControlEnum control_type, WWWFillInSpecifierEnum specifier, string description, string value);

boolean Fill_In (WWWControlEnum control_type, integer count, WWWFillInSpecifierEnum specifier, string description, string value);

boolean Fill_In (WWWControlEnum control_type, integer count, string value);

Get

Applies to Visual Scripting. Retrieves data from the virtual browser.

Versions

Versions for Get are:

page_id Get (WWWGetTypeEnum type);

page_id Get (WWWGetTypeEnum type, string description);

page_id Get (WWWGetTypeEnum type, string description, integer count);

page_id Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, string description);

page_id Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, string description, integer count);

page_id Get (WWWGetTypeEnum type, integer count);

integer Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, string left, string right);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, integer count, string left, string right);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, string xpath-string);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, string description, integer count);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, integer count);

string Get (WWWGetTypeEnum type, WWWGetSpecifierEnum specifier, integer form index, string description, integer count)

GetSiebelValue

Retrieves the value of a Siebel record from the Siebel library and saves it in a Local Variable.

A call to SiebelUpdatePage before the last request is mandatory to get the value of the Siebel record.

Syntax

```
GetSiebelValues(const char* pszSiebelUniqueID);
```

Return Value

CLoadString

Parameters

Parameter	Description
pszSiebelUniqueID	Unique Record ID of the Siebel parameter whose value is to be retrieved.

Example

```
CLoadString var = GetSiebelValue(" S_BC0_RC1_F01" );
```

ModifyEncoding

ModifyEncoding is used in Visual scripts to convert strings to UTF8, EUCJP or to the language used by the script.

Syntax

char* Modify_Encoding(EncodingLangEnum *encodingID*, const char* *strInput*)

Return Value

A char pointer to the encoded string if successful; NULL if not successful.

Parameters

Parameter	Description
encodingID	*EncodingTypeEnum* Counter data type. Valid values are: Value — Description UTF8 — UTF8 encoding. EUCJP — EUCJP encoding. SCRIPT_LANGUAGE — Script language encoding.
strInput	The input string.

Example

```
Fill_In(TEXT_BOX,NAME_ATTRIBUTE, "q", ModifyEncoding(SCRIPT_LANGUAGE,"Sample text to be converted to UTF-8"));
```

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.

Versions

Versions of Navigate_To are:

boolean Navigate_To (string URL);

boolean Navigate_To (string URL, WWWNavigateEncodingEnum encoding);

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 QALoad Installation and Configuration Guide.

Syntax

```
PlayMedia( int timeout );
```

Return Value

Parameters

Parameter	Description
timeout	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. Note: This timeout refers to clip time. The elapsed time of a Real Networks media transaction will likely be longer than the timeout.

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.

Versions

Versions of Post_To are:

boolean Post_To (string URL);

boolean Post_To (string URL, WWWPostContentTypeEnum content-type);

boolean Post_To (stringURL, WWWPostContentEncodingEnum encoding);

RandNumString

Applies to Visual Scripting. Generates a random number from minimum to maximum.

Syntax

```
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.

Syntax

```
string Region ( int region_number );
```

Return Value

Returns the region number as a string.

Parameters

Parameter	Description
region_number	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 first executes 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() ;
```

Return Value

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.

Restart_Transaction_Top is used if the transaction fails and Restart Transaction error handling has been selected in the QALoad Conductor.

Syntax

```
RESTART_TRANSACTION_TOP() ;
```

Return Value

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
```

Versions

Versions of Set are:

boolean Set (WWWSetDurationEnum
duration, WWWSetOptionBoolEnum
bool_option, boolean
boolean);

boolean Set (WWWSetDurationEnum
duration, WWWSetCachingOptionsEnum
cache_option, WWWSetCachingValuesEnum
cache_value);

```
boolean Set ( WWWSetDurationEnum duration,
                    WWWSetOptionIntegerEnum int_option,
                   integer integer );

boolean Set ( WWWSetDurationEnum
                   duration, WWWSetProxyOptionsEnum
                   proxy_option, WWWSetProxyModeValueEnum
                   proxy_mode_value );
boolean Set ( WWWSetDurationEnum duration,
                    WWWSetOptionTextEnum string1_option, string
                    string );
boolean Set ( WWWSetDurationEnum duration,
                    WWWSetOptionText2Enum string2_option, string
                    string1, string string2 );
boolean Set ( WWWSetDurationEnum
              duration, WWWSetOptionText3Enum
              string3_option, string
              string1, string
              string2, string
              string3 );
boolean Set ( WWWSetDurationEnum
              duration, WWWSetOptionNumericReferenceEnum int_option,
              NUMERIC_REFERENCE_LIST myReferences );
boolean Set ( WWWSetDurationEnum
              duration, WWWSetOptionEntityListEnum int_option,
              ENTITY_LIST myEntities );
boolean Set ( WWWSetDurationEnum
              duration, WWWSetOptionBoolEnum bool_option,
              WWWSetJavaScriptLevelEnum js_level );
```

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. 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 *QA*Load Player main window, in the Run As: group, select the Process option.
- Real Networks streaming media is only supported on a stand-alone *QA*Load Player or if the *QA*Load Player and *QA*Load Conductor are on the same machine.
- For streaming media playback, *QA*Load requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing *QA*Load" chapter of the *QALoad Installation and Configuration Guide*.

Syntax

```
ShowMediaRP( BOOL showAudio, BOOL showVideo );
```

Return Value

Parameters

Parameter	Description
showAudio	Display and play audio.
showVideo	Display video.

Example

```
ShowMediaRP( FALSE, TRUE );
// Display video, but leave audio muted
```

SiebelInitialize

Initializes the Siebel correlation library and treats the current script file as a Siebel file. If the Siebel correlation library dll (ssdtcorr.dll) is not present in the BinaryFiles folder under the QALoad directory, it throws an error in the TRACE window and exits the playback.

Syntax

```
SiebelInitialize()
```

Return Value

void

Parameters

None.

Example

```
SiebelInitialize()
```

SiebelUpdatePage

Feeds the current response (response from last request) to the Siebel library. This tells the Siebel library to extract the Siebel parameter values from the response.

Syntax

```
SiebelUpdatePage()
```

Return Value

void

Parameters

None.

Example

```
SiebelUpdatePage()
```

Verify

Applies to Visual Scripting. Used to verify expected text against an element of the page just requested.

Versions

Versions of Verify are:

boolean Verify (WWWVerificationSpecifierEnum type, string expected);

boolean Verify (WWWVerificationTypeEnum type, WWWVerificationSpecifierEnum specifier, string expected);

boolean Verify (WWWVerificationTypeEnum type, int count, WWWVerificationSpecifierEnum specifier, const char* description, Verify_Size size, string expected);

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 aborts 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 restarts 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 continues 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 ) ;
```

Return Value

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.

Syntax

```
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.

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 contains XML.

Syntax

```
boolean XMLRequest ( string method, string URL );
```

Return Value

True if the requested page is successfully retrieved.
False if the requested page is not successfully retrieved.

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://mssoapsampleserver/

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.

Syntax

```
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");
```

Index