QALoad 5.5 Help

Language Reference Commands, Error Codes, Customer Support, Accessibility, and Glossary

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

Customer Support Hotline:
1-800-538-7822

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

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

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

© 1998-2006 Compuware Corporation. All rights reserved. Unpublished - rights reserved under the Copyright Laws of the United States.

U.S. GOVERNMENT RIGHTS

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

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

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

Acrobat® Reader copyright © 1987-2006 Adobe Systems Incorporated. All rights reserved. Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.

ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright (c) 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

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

US Patent Nos.: Not Applicable.

Table Of Contents

Language Reference Commands 1
Overview 1
ADO 1
Citrix 176
ODBC 207
General Oracle 253
Oracle OCI Version 7 256
Oracle OCI Version 8 277
Oracle Forms Server 314
QALoad 390
SAP 4.x 422
SAP 6.x 436
SSL 452
Uniface 458
Winsock 485
WWW 517
Error Codes 588
Citrix 588
Oracle Forms Server 614
SAP 625
Winsock 635
WWW 673
Customer Support 720
Compuware customer support 720
Overview of the ZipFile Wizard 721
Accessibility 722
Accessibility features 722
Glossary 727
QALoad glossary 727
Index 733

Language Reference Commands

Overview

Contents of QALoad Language Reference

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:

ADO
Citrix
ODBC
Oracle OCI Version 7
General Oracle
Oracle OCI Version 8
Oracle Forms Server
QALoad
QARun Integration
SAP Versions 4.x
SAP Versions 6.20/6.40
SSL
Uniface
Polyserver
Winsock
WWW

ADO

ADO Commands

ADO_Command(n)->Cancel
Terminates the execution of an asynchronous method call.

ADO_Command(n)->CreateParameter
Creates a new Parameter object with a specified name, type, direction, size, and value. Any values passed in the arguments are written to the corresponding Parameter properties. This method does not automatically append the Parameter object to the Parameters collection of a Command object.

ADO_Command(n)->Execute
Executes the query specified in the CommandText property or CommandStream property of the object.

ADO_Command(n)->GetCommandStream
Retrieves the value contained in the CommandStream property of this instance of the ADO Command object. The CommandStream property is retrieved using a pointer to a variant.

ADO_Command(n)->GetCommandText
Retrieves the value of the CommandText property for this instance of the ADO Command object. A string is returned as its argument.

ADO_Command(n)->GetCommandTimeout
Retrieves the value contained within the CommandTimeout property of this instance of the ADO Command object.

ADO_Command(n)->GetCommandType
Retrieves the value contained in the CommandType property of the current instance of the Command object.

ADO_Command(n)->GetDialect
Retrieves the value contained within the Dialect property of this instance of the ADO Command object.

ADO_Command(n)->GetName
Allows the script to retrieve the value of the Name property for this instance of the ADO Command object.

ADO_Command(n)->GetNamedParameters
Retrieves the NamedParameters property of the Command object.

ADO_Command(n)->GetParameters
Retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

ADO_Command(n)->GetPrepared
Retrieves the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

ADO_Command(n)->GetProperties
Retrieves the complete set of properties for this particular instance of the Command object. PropertySets may change for different providers.

ADO_Command(n)->PutActiveConnection
Determines the Connection object affected by the specified Command object or ADO Recordset.

ADO_Command(n)->PutCommandText
Sets the value of the CommandText property for this instance of the ADO command object.

ADO_Command(n)->PutCommandTimeout
Sets the value contained within the CommandTimeout property of this instance of the ADO Command object.

ADO_Command(n)->PutCommandType
Sets the value for the CommandType property of the current instance of the Command object.

ADO_Command(n)->PutDialect
Sets the value of the Dialect property of this instance of the ADO Command object.

ADO_Command(n)->PutName
Enables the script to set the value of the Name property for this instance of the ADO Command object.

ADO_Command(n)->PutNamedParameters
Sets the value of the NamedParameters property of the command object.

ADO_Command(n)->PutPrepared
The PutPrepared method call sets the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

ADO_Command(n)->PutRefActiveConnection
Determines the ADO Connect object affected by the specified ADO Command object or ADO Recordset. Also sets the pointer to the QALoad ADO Connect object for this instance of the actual ADO Command object.

ADO_Connect(n)->BeginTrans
Begins a new transaction.

ADO_Connect(n)->Close
Closes a Connection object.

ADO_Connect(n)->CommitTrans
Save changes, ends the transaction. May start a new transaction.

ADO_Connect(n)->Execute
Executes the query passed in the CommandText argument on the connection to the method.

ADO_Connect(n)->GetAttributes
GetAttributes is read/write. It's value is the sum of one or more XactAttributeEnum values. The default is zero (0).

ADO_Connect(n)->GetCommandTimeout
Returns the value of the timeout in a pointer to a long.

ADO_Connect(n)->GetConnectionString
GetConnectionString method retrieves the value of ConnectionString property of this instance of the ADO Connect object. The ConnectionString specifies a data source by passing argument=value statements. These are separated by semi-colons.

ADO_Connect(n)->GetConnectionTimeout
Use GetConnectionTimeout on a Connection object to cancel a connection attempt, if necessary, due to network or server delays. If the time interval specified in the Connection Timeout property setting runs out before a connection can be opened, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

ADO_Connect(n)->GetCursorLocation
Lets you choose a cursor location from those accessible to the provider.

ADO_Connect(n)->GetDefaultDatabase
Retrieves the value of the DefaultDatabase property from this instance of the ADO Connect object.

ADO_Connect(n)->GetIsolationLevel
Sets a Connection object's isolation level. Takes effect the next time the BeginTrans method is called.

ADO_Connect(n)->GetMode
Sets or returns access permissions for the current connection. The GetMode property can only be set after the Connection object is closed.

ADO_Connect(n)->GetProvider
Returns the provider name for a connection.

ADO_Connect(n)->GetState
Determines the state of a specified object at any time.

ADO_Connect(n)->GetVersion
Returns the version of ADO that is being used.

ADO_Connect(n)->Open
Establishes the connection to the data source.

ADO_Connect(n)->OpenSchema
Returns information about the data source. For example, tables, columns included in the tables, data types, etc.

ADO_Connect(n)->PutAttributes
Sets the transaction attribute for this connection object.

ADO_Connect(n)->PutCommandTimeout
PutCommandTimeout Sends a timeout in seconds before the command will timeout with an error.

ADO_Connect(n)->PutConnectionString
Specifies a data source

ADO_Connect(n)->PutConnectionTimeout
Use PutConnectionTimeout on a Connection object to abandon an attempt to connect due to network or server delays. If a connection is not made in the specified time, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

ADO_Connect(n)->PutCursorLocation
Lets you choose a cursor library from those accessible to the provider.

ADO_Connect(n)->PutDefaultDatabase
Sets the default database within a connection object.

ADO_Connect(n)->PutIsolationLevel
Sets the isolation level of a Connection object.

ADO_Connect(n)->PutMode
Sets the access permissions being used on the current connection.

ADO_Connect(n)->PutProvider
Sets the provider name for a connection.

ADO_Connect(n)->RollbackTrans
Reverses changes made in an open transaction and ends the transaction. This is linked with BeginTrans. This will only be seen in the script if a transaction fails for some reason. If it fails and you see this call, look over the script logic and see if the transaction can be committed.

ADO_Field(n)->AppendChunk
A special data handling method that writes data, in chunks, to the Field object.

ADO_Field(n)->GetActualSize
Retrieves the value contained within the ActualSize property of this instance of the ADO Field object.

ADO_Field(n)->GetAttributes
Retrieves the value contained within the Attributes property of this instance of the ADO Field object.

ADO_Field(n)->GetChunk
Retrieves chunks of binary or character data to an appropriate buffer.

ADO_Field(n)->GetDataFormat
Retrieves an IUnknown instance describing the data format for this field.

ADO_Field(n)->GetDefinedSize
Retrieves the value contained within the DefinedSize property of this instance of the ADO Field object.

ADO_Field(n)->GetName
Retrieves the value contained within the Name property of this instance of the ADO Field object. Note that the actual ADO call has a BSTR as the argument; therefore, there is some data conversion occurring within this call.

ADO_Field(n)->GetNumericScale
Retrieves the value contained within the NumericScale property of this instance of the ADO Field object.

ADO_Field(n)->GetOriginalValue
Retrieves the value contained within the Value property of this instance of the ADO Field object before any changes were made permanent by a call to an update method.

ADO_Field(n)->GetPrecision
Retrieves the value contained within the Precision property of this instance of the ADO Field object.

ADO_Field(n)->GetProperties
The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

ADO_Field(n)->GetStatus
Retrieves the value contained within the Status property of this instance of the ADO Field object. It takes a pointer to a long as an argument. Within this parameter, the Value of the status property of this instance of the Field object is returned.

ADO_Field(n)->GetType
Retrieves the value contained within the Type property of this instance of the ADO Field object. The Actual

ADO call uses another enumerated type, DataTypeEnum, to handle the datatypes. Conversion to this type happens within the QALoad call.

ADO_Field(n)->GetUnderlyingValue
Specifies the current value of a Field object in the database, after any updates to the recordset.

ADO_Field(n)->GetValue
Retrieves the value of a ADO Field object into a pointer to a Variant. The argument will handle any type of data. This can be done by using the Variant datatype. Data is retrieved into a pointer to a Variant.

ADO_Field(n)->PutAttributes
The PutAttributes method call sets the value contained within the Attributes property of this instance of the ADO Field object.

ADO_Field(n)->PutDefinedSize
Sets the value contained within the DefinedSize property of this instance of the ADO Field object.

ADO_Field(n)->PutNumericScale
Sets the value contained within the NumericScale property of this instance of the ADO Field object.

ADO_Field(n)->PutPrecision
Sets the value contained within the Precision property of this instance of the ADO Field object.

ADO_Field(n)->PutRefDataFormat
This updates the current value of the data format updates to the ADO Recordset.

ADO_Field(n)->PutType
Sets the value contained within the Type property of this instance of the ADO Field object.

ADO_Field(n)->PutValue
Resets the value of this instance of the ADO Field object. This is the first step in updating a Recordset's value.

ADO_FieldSet(n)->Append
Append creates and appends a new Field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprises a mechanism for updating or retrieving information from a Data Provider.

ADO_FieldSet(n)->Append15
Append15 creates and appends a new field object to the ADO FieldSet. An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprise a mechanism for updating or retrieving information from a Data Provider. The Append15 function does NOT allow the user to add the data to this ADO Field object. It creates the ADO Field object in the ADO FieldSet collection, but does not add in the data.

ADO_FieldSet(n)->CancelUpdate
Cancels changes made to the current or new row of an ADO Recordset object, or the ADO Fieldset collection of an ADO Record object, before calling the Update method.

ADO_FieldSet(n)->Delete
Deletes an object from the Fields collection.

ADO_FieldSet(n)->GetCount
The method returns the number of ADO Field objects contained within the ADO FieldSet collection.

ADO_FieldSet(n)->GetItem
This call retrieves a ADO Field object from this instance of the ADO FieldSet collection. The result of the call is that a ADO Field object is brought back to be manipulated within the script. ADO Field retrieval is a part of the variablization process.

ADO_FieldSet(n)->GetNewEnum
Creates the ADO IEnumField object.

ADO_FieldSet(n)->Refresh
Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO FieldSet collection has no visible effect.

ADO_FieldSet(n)->Resync
Synchronizes the values of a Record object's Fields collection with the data source. The Count property is not affected by this method.

ADO_FieldSet(n)->Update
Saves any changes you make to the ADO FieldSet collection of a Record object.

ADO_IEnum(n)->NextProperty
Enumeration through collections of properties should be done very carefully, because in the example below, we will reset all of the properties to the same value. To reset different values, get rid of the loop and set each property individually.

ADO_IEnumField(n)->NextField
Enumeration through collections of ADO Fields should be done very carefully, because in the example given below, the script checks the status of each of the different ADO Fields. In order to do more meaningful work, reset values of different ADO Fields then break them out of the look and use the PutValue call to place new values into the ADO Field objects.

ADO_IEnumParameter(n)->NextParameter
Enumeration through collections of ADO Parameters should be done very carefully, because in the example given below, the script checks different values of each of the different ADO Parameters. In order to do some more meaningful work, resetting values of different ADO Parameters then break them out of the look and use the PutValue call to place new values into the ADO Parameter objects.

ADO_LoadVariant(n)
Loads the value sValue, of type sType, into the Variant structure.

ADO_Parameter(n)->AppendChunk
A special data handling method that writes data, in chunks, to the Parameter object.

ADO_Parameter(n)->GetAttributes
Retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

ADO_Parameter(n)->GetDirection
Retrieves the value contained within the Direction property of this instance of the ADO Parameter object.

ADO_Parameter(n)->GetName
Retrieves the value contained within the Name property of this instance of the ADO Parameter object.

ADO_Parameter(n)->GetNumericScale
Retrieves the value contained within the NumericScale property of this instance of the ADO Parameter object. Returns a Byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

ADO_Parameter(n)->GetPrecision
Retrieves the value contained within the Precision property of this instance of the ADO Parameter object. Returns a Byte value showing the maximum number of digits used to represent values for a numeric Parameter object. The Precision property is read/write.

ADO_Parameter(n)->GetSize
Retrieves the value contained within the Size property of this instance of the ADO Field object.

ADO_Parameter(n)->GetValue
Use the Value property to return data from ADO Parameter objects and to return parameter values with ADO Parameter objects.

ADO_Parameter(n)->PutAttributes
This method call retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutDirection
Indicates Parameter type: input, output, input and output, or the return value from a stored procedure.

ADO_Parameter(n)->PutName
Sets the value contained within the Name property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutNumericScale
Sets the value contained within the NumericScale property of this instance of the ADO Parameter object. Sends a byte value indicating the number of decimal places to which numeric values will be resolved. The NumericScale property is read/write.

ADO_Parameter(n)->PutPrecision
Sets the value contained within the Precision property of this instance of the ADO Parameter object. Sends a byte value showing the maximum number of digits used to represent values for a numeric ADO Parameter object. The Precision property is read/ write.

ADO_Parameter(n)->PutSize
Retrieves the value contained within the Size property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutType
Sets the value contained within the Type property of this instance of the ADO Parameter object.

ADO_Parameter(n)->PutValue
Sets the value contained within the Value property of this instance of the ADO Parameter object.

ADO_ParameterSet(n)->Append
Appends a ADO Parameter object to the collection of ADO Parameters.

ADO_ParameterSet(n)->Delete
Deletes an ADO Parameter object from the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetCount
The method returns the number of ADO Parameter objects contained within the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetItem
Locates a specific ADO Parameter in the ADO ParameterSet collection.

ADO_ParameterSet(n)->GetNewEnum
In order to iterate through all of the ADO Parameters in an ADO ParameterSet collection, an ADOIEnumParameter object is returned. The GetNewEnum call on the ADO ParameterSet object creates the ADO IEnumParameter object allowing the enumeration to take place.

ADO_ParameterSet(n)->Refresh
Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO ParameterSet collection has no visible effect.

ADO_Property(n)->GetAttributes
Describes column characteristics by setting or returning a Long value.

ADO_Property(n)->GetName
Retrieves the value of the Name attribute of this instance of the Property object.

ADO_Property(n)->GetType
Indicates a property's type as conveyed as a DataTypeEnum.

ADO_Property(n)->GetValue
Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

ADO_Property(n)->PutAttributes
Sets the value contained within the Attributes property of this instance of the ADO Property object.

ADO_Property(n)->PutValue
Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

ADO_PropertySet(n)->GetCount
The method returns the number of ADO Property objects contained within the ADO PropertySet collection.

ADO_PropertySet(n)->GetItem
Retrieves a specific ADO Property in the ADO PropertySet collection.

ADO_PropertySet(n)->GetNewEnum
In order to iterate through all of the ADO Propertys in an ADO PropertySet collection, an ADOIEnum object is returned. The GetNewEnum call on the ADO PropertySet object creates the ADO IEnum object allowing the enumeration to take place.

ADO_PropertySet(n)->Refresh
Updates the objects in a collection to reflect objects available from, and specific to, the provider. Using the Refresh method on the ADO PropertySet collection has no visible effect.

ADO_Record(n)->Cancel
Cancels execution of a pending, asynchronous method call.

ADO_Record(n)->Close
Use to close a Recordset, Record, or Stream object. Any associated data or exclusive access you may have had to the data through this particular object will be released. You can reopen the object later using the Open method.

ADO_Record(n)->CopyRecord
Copies a file or directory (including its contents) to another location.

ADO_Record(n)->DeleteRecord
Deletes a file or directory, and all its subdirectories.

ADO_Record(n)->GetActiveConnection
Use the ActiveConnection property to determine the ADO Connect object over which the specified ADO Record object will execute.

ADO_Record(n)->GetChildren
Returns an ADO Recordset, in the form of a Pointer to an ADO Recordset object, whose rows represent the files and subdirectories in the directory represented by this Record.

ADO_Record(n)->GetFields
Contains all the Field objects of an ADO Recordset or ADO Record object.

ADO_Record(n)->GetMode
Sets or returns the access permissions being used on the current connection by the provider.

ADO_Record(n)->GetParentURL
Sets the current value of the source property for this instance of the actual ADO Command object.

ADO_Record(n)->GetRecordType
This method is used to check the contents of the ADO RecordType property for this instance of ADO Record object, returning the RecordTypeEnum in a pointer to a long.

ADO_Record(n)->GetSource
Indicates the entity represented by the ADO Record object.

ADO_Record(n)->GetState
You can use the State property to determine the state of a given ADO Record object at any time.

ADO_Record(n)->MoveRecord
Moves a file, or a directory and its contents, to another location.

ADO_Record(n)->Open
Makes the call through to the Open method within the ADO Record object to open an existing ADO Record object, or create a new file or directory.

ADO_Record(n)->PutActiveConnection
PutActiveConnection is read/write when the ADO Record object is closed. It may contain a connection string or reference to an open ADO Connect object. When the ADO Record object is open and contains a reference to an open ADO Connect object, PutActiveConnection is read-only.

ADO_Record(n)->PutMode
Sets the access permissions being used on the current connection by the provider. You can only set this property when the ADO Connect object is closed.

ADO_Record(n)->PutRefActiveConnection
Specifies the ADO Connect object to be affected by the specified ADO Record object.

ADO_Record(n)->PutRefSource
Sets a Command object as the data source for a Recordset object.

ADO_Record(n)->PutSource
Sets the current value of the source property for this instance of the actual ADO Command object. The Source property must refer to an object existing within the scope of that ADO Connect.

ADO_Recordset(n)->_xClone
This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it makes a clone of the calling ADO Recordset. This is given the arguments and the method name.

ADO_Recordset(n)->_xResync
This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it resynchronizes the ADO Recordset with the underlying data provider. This is given the arguments and the method name.

ADO_Recordset(n)->_xSave
This is a hidden method. It is undocumented within MSDN. However, a logical assumption would be that it saves ADO Recordset data to the location given in the first argument. This is given the arguments and the method name.

ADO_Recordset(n)->AddNew
Creates a new record for an updatable ADO Recordset object.

ADO_Recordset(n)->Cancel
Cancels execution of a pending, asynchronous method call.

ADO_Recordset(n)->CancelBatch
Cancels any pending updates in an ADO Recordset that is in batch update mode.

ADO_Recordset(n)->CancelUpdate
Cancels any changes made to the current row or discards a new row of an ADO Recordset object before calling the Update method.

ADO_Recordset(n)->Clone
Duplicates an ADO Recordset object. Can specify that the clone be read-only.

ADO_Recordset(n)->Close
Closes an open object and any dependent objects.

ADO_Recordset(n)->CompareBookmarks
Compares two bookmarks. Returns an indication of their relative values.

ADO_Recordset(n)->Delete
Use to delete the current record or a group of records.

ADO_Recordset(n)->Find
Locates a row in an ADO Recordset that matches specified criteria.

ADO_Recordset(n)->GetAbsolutePage
Identifies, by page number, where the current record resides.

ADO_Recordset(n)->GetAbsolutePosition
Specifies the ordinal position of the current record of an ADO Recordset object.

ADO_Recordset(n)->GetActiveCommand
Specifies the ADO Command object which created an ADO Recordset object.

ADO_Recordset(n)->GetActiveConnection
For a Command, ADO Recordset, or ADO Record object, specifies the associated ADO Connect object.

ADO_Recordset(n)->GetBOF
Determines if an ADO Recordset object contains records or if you've gone beyond its limits while moving from record to record.

ADO_Recordset(n)->GetBookmark
Indicates a bookmark identifying an ADO Recordset object's current record, or sets the current record to that identified by a bookmark.

ADO_Recordset(n)->GetCacheSize
Specifies the number of records in the ADO Recordset that are cached locally.

ADO_Recordset(n)->GetCollect
This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

ADO_Recordset(n)->GetCursorLocation
Specifies the location of the cursor service.

ADO_Recordset(n)->GetCursorType
Specifies the type of cursor to use when opening the ADO Recordset object.

ADO_Recordset(n)->GetDataMember
Specifies the data member to be retrieved from the object referenced by the DataSource property.

ADO_Recordset(n)->GetDataSource
Specifies an object containing data to be represented as an ADO Recordset object.

ADO_Recordset(n)->GetEditMode
Specifies the current record's editing status.

ADO_Recordset(n)->GetEOF
Indicates that the current record position is after the last record in an ADO Recordset object.

ADO_Recordset(n)->GetFields
Returns a container of an ADO Recordset or ADO Record object's Field objects.

ADO_Recordset(n)->GetFilter
Specifies a filter for data in an ADO Recordset.

ADO_Recordset(n)->GetIndex
This is a hidden method. It is undocumented within MSDN.

ADO_Recordset(n)->GetLockType
Specifies the type of locks placed on records during editing.

ADO_Recordset(n)->GetMarshalOptions
Specifies records to be marshaled back to the server.

ADO_Recordset(n)->GetMaxRecords
Specifies the maximum number of records to return to an ADO Recordset from a query.

ADO_Recordset(n)->GetPageCount
Specifies the number of pages of data contained in the ADO Recordset object.

ADO_Recordset(n)->GetPageSize
Indicates the number of records that make up a single page in the ADO Recordset.

ADO_Recordset(n)->GetProperties
The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

ADO_Recordset(n)->GetRecordCount
Indicates the number of records in an ADO Recordset object.

ADO_Recordset(n)->GetRows
Retrieves multiple records of an ADO Recordset object into an array.

ADO_Recordset(n)->GetSort
Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

ADO_Recordset(n)->GetSource
Indicates the data source for a Recordset object.

ADO_Recordset(n)->GetState
Indicates for all applicable objects whether the state of the object is open or closed.

ADO_Recordset(n)->GetStatus
Indicates the status of the current record with respect to batch updates or other bulk operations.

ADO_Recordset(n)->GetStayInSync
Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

ADO_Recordset(n)->GetString
Returns the ADO Recordset as a string.

ADO_Recordset(n)->Move
Moves the position of the current record in an ADO Recordset object.

ADO_Recordset(n)->MoveFirst
Use the MoveFirst method to move the current record position to the first record in the ADO Recordset.

ADO_Recordset(n)->MoveLast
Use the MoveLast method to move the current record position to the last record in the ADO Recordset. The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error.

ADO_Recordset(n)->MoveNext
Use the MoveNext method to move the current record position one record forward (toward the bottom of the ADO Recordset). If the last record is the current record and you call the MoveNext method, ADO sets the current record to the position after the last record in the ADO Recordset (EOF is True). An attempt to move forward when the EOF property is already True generates an error.

ADO_Recordset(n)->MovePrevious
Use the MovePrevious method to move the current record position one record backward (toward the top of the ADO Recordset). The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call will generate an error. If the first record is the current record and you call the MovePrevious method, ADO sets the current record to the position before the first record in the ADO Recordset (BOF is True).

ADO_Recordset(n)->NextRecordset
Clears the current ADO Recordset object and returns the next ADO Recordset by advancing through a series of commands.

ADO_Recordset(n)->Open
Using the Open method on an ADO Recordset object opens a cursor that represents records from a base table, the results of a query, or a previously saved ADO Recordset.

ADO_Recordset(n)->PutAbsolutePage
Indicates on which page the current record resides.

ADO_Recordset(n)->PutAbsolutePosition
Indicates the ordinal position of an ADO Recordset object's current record.

ADO_Recordset(n)->PutActiveConnection
Indicates to which Connection object the specified Command, ADO Recordset, or Record object currently belongs.

ADO_Recordset(n)->PutBookmark
Indicates a bookmark that uniquely identifies the current record in an ADO Recordset object or sets the current record in an ADO Recordset object to the record identified by a valid bookmark.

ADO_Recordset(n)->PutCacheSize
Indicates the number of records in the ADO Recordset that are cached locally.

ADO_Recordset(n)->PutCollect
This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit. Neither QALoad support professionals nor development recommend adding this method to a script.

ADO_Recordset(n)->PutCursorLocation
Indicates the location of the cursor service.

ADO_Recordset(n)->PutCursorType
Use the CursorType property to specify the type of cursor that should be used when opening the ADO Recordset object.

ADO_Recordset(n)->PutDataMember
Indicates the name of the data member that will be retrieved from the object referenced by the DataSource property.

ADO_Recordset(n)->PutFilter
Indicates a filter for data in an ADO Recordset.

ADO_Recordset(n)->PutIndex
This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below and do as you see fit.

ADO_Recordset(n)->PutLockType
Indicates the type of locks placed on records during editing.

ADO_Recordset(n)->PutMarshalOptions
Indicates which records are to be marshaled back to the server.

ADO_Recordset(n)->PutMaxRecords
Indicates the maximum number of records to return to an ADO Recordset from a query.

ADO_Recordset(n)->PutPageSize
Indicates how many records constitute one page in the ADO Recordset.

ADO_Recordset(n)->PutRefActiveConnection
Indicates to which ADO Connect object the specified ADO Command, ADO Recordset, or Record object currently belongs.

ADO_Recordset(n)->PutRefDataSource
Indicates an object that contains data to be represented as an ADO Recordset object.

ADO_Recordset(n)->PutRefSource
Sets a Command object as the data source for a Recordset object.

ADO_Recordset(n)->PutSort
Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

ADO_Recordset(n)->PutSource
Indicates the data source for an ADO Recordset object.

ADO_Recordset(n)->PutStayInSync
Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

ADO_Recordset(n)->ReQuery
Updates the data in an ADO Recordset object by re-executing the query on which the object is based.

ADO_Recordset(n)->Resync
Refreshes the data in the current ADO Recordset object, or Fields collection of a Record object, from the underlying database.

ADO_Recordset(n)->Save
Saves the ADO Recordset in a file or ADO Stream object.

ADO_Recordset(n)->Seek
The SeekEnum is an Enumerated value giving the direction of the seek operation.

ADO_Recordset(n)->Supports
Determines whether a specified ADO Recordset object supports a particular type of functionality.

ADO_Recordset(n)->Update
Saves any changes you make to the current row of an ADO Recordset object.

ADO_Recordset(n)->UpdateBatch
Writes all pending batch updates within the ADO Recordset to disk.

ADO_Stream(n)->Cancel
Cancels execution of a pending ADO Stream, asynchronous method call.

ADO_Stream(n)->Close
Closes an open object and any dependent objects.

ADO_Stream(n)->CopyTo
Copies the specified number of characters or bytes (depending on Type) in the ADO Stream to another ADO Stream object.

ADO_Stream(n)->Flush
Forces the contents of the ADO Stream remaining in the ADO buffer to the underlying object with which the ADO Stream is associated.

ADO_Stream(n)->GetCharset
Indicates the character set into which the contents of a text ADO Stream should be translated.

ADO_Stream(n)->GetEOS
Indicates whether the current position is at the end of the ADO Stream.

ADO_Stream(n)->GetLineSeparator
Indicates the binary character to be used as the line separator in text ADO Stream objects.

ADO_Stream(n)->GetMode
Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

ADO_Stream(n)->GetPosition
Indicates the current position within an ADO Stream object.

ADO_Stream(n)->GetSize
Returns a Long value that specifies the size of the ADO Stream in number of bytes. The default value is the size of the ADO Stream, or -1 if the size of the ADO Stream is not known.

ADO_Stream(n)->GetState
The ADO Stream object's State property can have a combination of values. For example, if a statement is executing, this property will have a combined value of adStateOpen and adStateExecuting.

ADO_Stream(n)->GetType
Indicates the type of data contained in the ADO Stream (binary or text).

ADO_Stream(n)->LoadFromFile
Loads the contents of an existing file into an ADO Stream.

ADO_Stream(n)->PutCharset
Indicates the character set into which the contents of a text ADO Stream should be translated.

ADO_Stream(n)->PutLineSeparator
Indicates the binary character to be used as the line separator in text ADO Stream objects.

ADO_Stream(n)->PutMode
Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

ADO_Stream(n)->PutPosition
Indicates the current position within an ADO Stream object.

ADO_Stream(n)->PutType
Indicates the type of data contained in the ADO Stream (binary or text).

ADO_Stream(n)->Read
Reads a specified number of bytes from a binary ADO Stream object.

ADO_Stream(n)->ReadText
Reads specified number of characters from a text ADO Stream object.

ADO_Stream(n)->SaveToFile
Saves the number of bytes contents of the current ADO Stream to the file from the current position. It sends the second param number of bytes to that File.

ADO_Stream(n)->SetEOS
Sets the current position within the ADO Stream as the End of the ADO Stream

ADO_Stream(n)->SkipLine
Skips one entire line when reading a text ADO Stream.

ADO_Stream(n)->Write
Writes BINARY Data to the ADO Stream buffer.

ADO_Stream(n)->WriteText
Writes a specified text string to an ADO Stream object.

ADOStream(n)->Open
Opens an ADO Stream object to manipulate streams of binary or text data.

ExtractVariantValue
Retrieves the contents of a variant and places that value in a string.

PrintVariant
Decodes variant data and places this data into a string.

ADO_Command(n)->Cancel

Terminates the execution of an asynchronous method call.

Syntax

```
ADO_Command(n)->Cancel();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Command(0)->CreateParameter( "ParamTest1", adEmpty, adParamInput, 0, pvValue,
ADOParameter[0] );
ADOParameter.Release( 0 );
ADO_LoadVariant( pvValue, "8", "A Test Value" );
ADO_Command(0)->CreateParameter( "TestParam2", adEmpty, adParamInputOutput, 100, pvValue,
ADOParameter[0] );
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->CreateParameter

Creates a new Parameter object with a specified name, type, direction, size, and value.

Any values passed in the arguments are written to the corresponding Parameter properties. This method does not automatically append the Parameter object to the Parameters collection of a Command object.

CreateParameter lets you set additional properties whose values ADO validate when appending the Parameter object to the collection. If specifying a variable length data type in the Type argument, you must either pass a Size argument, or set the Size <mdprosize.htm> property of the Parameter object before appending it to the Parameters collection. Otherwise, an error occurs.

If you specify a numeric data type (adNumeric or adDecimal) in the Type argument, then you must also set the NumericScale <mdpronumericscale.htm> and Precision <mdproprecision.htm> properties.

Syntax

```
ADO_Command(n)->CreateParameter( char* sName, ADODataTypeEnum Type,
ADOParameterDirectionEnum Direction, long nLength, VARIANT* pvValue, CAParameter* pParameter
);
```

Return Value

Parameters

Parameter	Description

n	Index to the object.
sName	String containing the name of the parameter.
Type	ADODataTypeEnum Enumerated type describing the data type. Valid values are: Value — Description adEmpty — No data type specified adTinyInt — Tiny integer adSmallInt — Small integer adInteger — Integer adBigInt — Big integer adUnsignedTinyInt — Unsigned tiny integer adUnsignedSmallInt — Unsigned small integer adUnsignedInt — Unsigned integer adUnsignedBigInt — Unsigned integer adSingle — Single precision float adDouble — Double precision float adCurrency — Currency adDecimal — Decimal adNumeric — Numeric adBoolean — Boolean adError — Error adUserDefined — User-defined data type adVariant — Variant adIDispatch — Pointer to IDispatch adIUnknown — Pointer to IUnknown adGUID — GUID adDate — Date adDBDate — DBDate adDBTime — DBTime

	adDBTimeStamp — DBTimestamp adBSTR — BSTR adChar — Char adVarChar — VarChar adLongVarChar — Long VarChar adWChar — Wide Char adVarWChar — VarWChar adLongVarWChar — Long VarWChar adBinary — Binary adVarBinary — VarBinary adLongVarBinary — Long VarBinary adChapter — Chapter adFileTime — FileTime adPropVariant — Variant Property adVarNumeric — VarNumeric adArray — Array
Direction	ADOParameterDirectionEnum Enumerated type describing the direction of parameter. Valid values are: Value — Description adParamUnknown — Unknown parameter status adParamInput — Parameter is input value adParamOutput — Parameter is output value adParamInputOutput — Parameter is input/output value adParamReturnValue — Parameter is return value
nLength	Integer variable specifying the maximum length, in bytes, of the parameter.
pvValue	Value of parameter at time of creation.
pParameter	Created parameter.

Example

```
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdUnknown );
ADO_Command(0)->GetState( pLong );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Command(0)->CreateParameter( "ParamTest1", adEmpty, adParaminput, 0, pvValue,
ADOParameter[0] );
ADO_LoadVariant( pvValue, "8", "A Test Value" );
```

ADO_Command(n)->Execute

Executes the query specified in the CommandText property or CommandStream property of the object.

Syntax

```
ADO_Command(n)->Execute( VARIANT* pvRecNo, VARIANT*
pvParamList, long nCommandType, CARecordset* pRecordset );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvRecNo	A long variable to which the provider returns the number of records that the operation affected.
pvParamList	A variant array of parameter values passed.
nCommandType	A long value that indicates how the provider should evaluate the CommandText property of the Command object.
pRecordset	*QA*Load wrapper object, returned ADORecordset.

Example

```
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
ADO_LoadVariant( pvValue, "3", "-1" );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Command(0)->Execute( pvValue, pvSource, 1,
ADORecordset[0] );
```

ADO_Command(n)->GetCommandStream

Retrieves the value contained in the CommandStream property of this instance of the ADO Command object.

The CommandStream property is retrieved using a pointer to a variant.

Syntax

```
ADO_Command(n)->GetCommandStream( VARIANT* pvValue );
```

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a variant into which the data is retrieved.

Example

```
ADO_Command(0)->GetCommandStream( pvValue );
ADO_Command(0)->GetDialect( pvSource );
```

ADO_Command(n)->GetCommandText

Retrieves the value of the CommandText property for this instance of the ADO Command object. A string is returned as its argument.

The CommandText property is returned inside a CLoadString.

Syntax

```
ADO_Command(n)->GetCommandText( CLoadString& sCommandText );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sCommandText	The CommandText property for which a value is to be returned.

Example

```
ADO_Command(0)->PutCommandText("Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
ADO_Connect(0)>Open("PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa; pwd="";database=Master;", "",
"", -1 );
```

ADO_Command(n)->GetCommandTimeout

Retrieves the value contained within the CommandTimeout property of this instance of the ADO Command object.

Use CommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Command(n)->GetCommandTimeout( long* pCommandTimeout );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCommandTimeout	Pointer to a long containing the time interval.

Example

```
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 250 );
```

ADO_Command(n)->GetCommandType

Retrieves the value contained in the CommandType property of the current instance of the Command object.

GetCommandType should always be combined with adCmdText or adCmdStoredProc, for example, adCmdText+adExecuteNoRecords. Note that using adExecuteNoRecords with the Open method or a Command object used by that method returns an error.

Syntax

```
ADO_Command(n)->GetCommandType( long* plCommandType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
plCommandType	Pointer to a long containing a command type Enum.

Example

```
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->GetDialect

Retrieves the value contained within the Dialect property of this instance of the ADO Command object.

The Dialect property contains a valid GUID (Globally Unique Identifier) that represents the dialect of the command text or stream. The default value for this property is {C8B521FB-5CF3-11CE-ADE5-00AA0044773D}, indicating that the provider should choose how to interpret the command text or stream.

Syntax

```
ADO_Command(n)->GetDialect( CLoadString sDialect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sDialect	A CLoadString. Value of the Dialect property for this instance of the ADO Command.

Example

```
ADO_Command(0)->PutName( "MyTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutDialect( "{C8B521FB-5CF3-11CE-ADE5-00AA0044773D}" );
ADO_Command(0)->GetDialect( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
```

ADO_Command(n)->GetName

Allows the script to retrieve the value of the Name property for this instance of the ADO Command object.

Syntax

```
ADO_Command(n)->GetName( CLoadString& sName );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sName	A CLoadString. Value of the Name property for this instance of the ADO Command.

Example

```
ADOConnect.Release( 0 );
ADO_Command(0)->PutName( "aTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
```

ADO_Command(n)->GetNamedParameters

Retrieves the NamedParameters property of the Command object.

When true, the Command object handles the parameters by name instead of by order. The method of retrieving parameters depends on the value of the NamedParameters property.

Syntax

```
ADO_Command(n)->GetNamedParameters( VARIANT_BOOL* pnParameter );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pnParameter	A short evaluating to either True (-1) or False (0).

Example

```
ADO_Command(0)->PutCommandTimeout( 250 );
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetNamedParameters( pVTBOOL );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->GetParameters

Retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

A Command object has a collection of Parameters associated with it, holding zero or more Parameters within it. Using the Refresh method on a Command object's Parameters collection retrieves provider parameter information for the stored procedure or parameterized query specified in the Command object.

Note: The n associated with the ADO_Command object and that associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
ADO_Command(n)->GetParameters( CAParameterSet* pParameterSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pParameterSet	An instance of the ADO ParameterSet object. This instance is being

	created and filled as a result of this call.

Example

```
ADO_Command(0)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
ADO_Command(0)->PutCommandText( "op_Getparamvb6" );
BeginCheckpoint("ADOCommand::GetParameters");
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
EndCheckpoint("ADOCommand::GetParameters");
```

ADO_Command(n)->GetPrepared

Retrieves the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

A VARIANT_BOOL is a short evaluating to either True (-1) or False (0). As a result, the provider saves a compiled version of the query specified in the CommandText property before a Command object's first execution. This might slow down the initial execution; however, performance improves in subsequent executions because the provider uses the compiled version of the command.

Syntax

```
ADO_Command(n)->GetPrepared( short* pPrepared  );
```

Return Value

Parameters

Parameter	Description
n	An index to an object.
pPrepared	Pointer to the VARIANT_BOOL value contained in the Prepared property of this instance of the ADO Command object.

Example

```
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->GetProperties

Retrieves the complete set of properties for this particular instance of the Command object.

PropertySets may change for different providers.

Syntax

```
ADO_Command(n)->GetProperties( CAPropertySet* pPropertySet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pPropertySet	Set of CAProperty objects. Each CAProperty object contains a single characteristic, a piece of data, which partially describes the state of a particular instance of an object.

Example

```
ADO_LoadVariant( pvValue, "8", "test_number" );
ADO_Command(0)->GetItem( pvValue, ADOCommand[0] );
ADO_Command(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Command(n)->PutActiveConnection

Determines the Connection object affected by the specified Command object or ADO Recordset.

A call to LoadVariant must be made before each call to PutActiveConnection. This instance of LoadVariant takes ADO Connect information and loads that into a pointer to a variant.

Note: The n associated with the ADO_Command object and the n associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
ADO_Command(n)->PutActiveConnection( VARIANT* pvConnection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvConnection	Pointer to a variant containing the ADO Connection Object.

Example

```
ADO_Command(2)->PutCommandText( "sp_GetParameterSet" );
ADO_Command(2)->PutCommandType( adCmdStoredProc );
LoadVariant( pvValue, ADOConnect[0] );
ADO_Command(2)->PutActiveConnection( pvValue );
BeginCheckpoint("ADOCommand::GetParameters");
ADO_Command(2)->GetParameters( ADOParameterSet[0] );
EndCheckpoint("ADOCommand::GetParameters");
```

ADO_Command(n)->PutCommandText

Sets the value of the CommandText property for this instance of the ADO command object.

One method of setting up a command, for instance a SQL statement of some sort, would be by using PutCommandText.

Syntax

```
ADO_Command(n)->PutCommandText( char* sText );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sText	The ANSI value of the CommandText property.

Example

```
ADO_Command(0)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
BeginCheckpoint( "ADOCommand::GetParameters" );
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
EndCheckpoint( "ADOCommand::GetParameters" );
```

ADO_Command(n)->PutCommandTimeout

Sets the value contained within the CommandTimeout property of this instance of the ADO Command object.

Use CommandTimeout on a Connection or Command object to allow an Execute method call to be cancelled for network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Command(n)->PutCommandTimeout( long nCommandTimeout );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nCommandTimeout	Timeout value (a non-negative integer) to set for this instance of the ADO Command object.

Example

```
ADO_Connect(0)->PutConnectionString( "DSN=My;UID=sa; PWD=" ";" );
BeginCheckpoint("ADOConnect::Open");
ADO_Connect(0)->Open( "", "", "", -1 );
EndCheckpoint("ADOConnect::Open");
ADO_Command(0)->PutCommandTimeout( 200 );
```

ADO_Command(n)->PutCommandType

Sets the value for the CommandType property of the current instance of the Command object.

This is generally done using an enumerated type. In this case, the CommandTypeEnum would be used. Since all of the different enumerated types within ADO essentially boil down to longs, longs are accepted as the argument.

Caution: Randomly introducing numbers into the script for this operation has unpredictable results.

PutCommandType should always be combined with adCmdText or adCmdStoredProc, for example, adCmdText+adExecuteNoRecords. Using adExecuteNoRecords with the Open method, or a Command object used by that method, results in an error.

Syntax

```
ADO_Command(n)->PutCommandType ( ADOCommandTypeEnum adCmdText );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
adCmdText	ADOCommandTypeEnum The enumerated datatype identifies the type of Command. Valid values are: Value — Description adCmdUnspecified — Does not specify the command type argument adCmdUnknown — Default. Indicates that the type of command in the CommandText property is not known adCmdText — Evaluates CommandText as a textual definition of a command or stored procedure call adCmdTable — Evaluates CommandText as a table name whose columns are all returned by an internally generated SQL query adCmdStoredProc — Evaluates CommandText as a stored procedure name adCmdFile — Evaluates CommandText as the file name of a persistently stored Recordset adCmdTableDirect — Evaluates CommandText as a table name whose columns are all returned

Example

```
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetCommandType( pLong );
ADO_Command(0)->PutCommandType( adCmdStoredProc );
```

ADO_Command(n)->PutDialect

Sets the value of the Dialect property of this instance of the ADO Command object.

The Dialect property contains a valid GUID (Globally Unique Identifier) that represents the dialect of the command text or stream. The default value for this property is {C8B521FB-5CF3-11CE-ADE5-00AA0044773D}, which indicates that the provider should choose how to interpret the command text or stream.

When the Dialect property is set, ADO validates the GUID and raises an error if the value supplied is not a valid GUID. Refer to your provider documentation to determine the GUID values supported by the Dialect property.

Syntax

```
ADO_Command(n)->PutDialect( char* sDialect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sDialect	A unique identifier for a dialect.

Example

```
ADO_Command(0)->PutName( "MyTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutDialect( "{C8B521FB-5CF3-11CE-ADE5-00AA0044773D"}" );
ADO_Command(0)->GetDialect( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
ADO_Command(0)->GetCommandText( sLoadStr );
```

ADO_Command(n)->PutName

Enables the script to set the value of the Name property for this instance of the ADO Command object.

Syntax

```
ADO_Command(n)->PutName( char* sName );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sName	Value of the Name property for this instance of the ADO Command object.

Example

```
ADOConnect.Release( 0 );
ADO_Command(0)->PutName( "aTest" );
ADO_Command(0)->GetName( sLoadStr );
ADO_Command(0)->PutCommandText( "Select * from test_table" );
```

ADO_Command(n)->PutNamedParameters

Sets the value of the NamedParameters property of the command object.

When true, the Command object handles the parameters by name instead of by order. The method of retrieving parameters depends on the value of the NamedParameters property.

Syntax

```
ADO_Command(n)->PutNamedParameters( short nParameter );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
short nParameter	A short evaluating to either True (-1) or False (0).

Example

```
ADO_Command(0)->GetCommandTimeout( pLong );
ADO_Command(0)->PutCommandTimeout( 30 );
ADO_Command(0)->GetNamedParameters( pVTBOOL );
ADO_Command(0)->PutNamedParameters( 0 );
```

ADO_Command(n)->PutPrepared

The PutPrepared method call sets the VARIANT_BOOL value contained within the Prepared property of this instance of the ADO Command object.

A VARIANT_BOOL is a short evaluating to either True (-1) or False (0).

Syntax

```
ADO_Command(n)->PutPrepared( short flag );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
flag	TRUE (-1) or FALSE (0).

Example

```
ADOParameter.Release( 0 );
ADO_Command(0)->Cancel();
ADO_Command(0)->PutPrepared( -1 );
```

ADO_Command(n)->PutRefActiveConnection

Determines the ADO Connect object affected by the specified ADO Command object or ADO Recordset.

It also sets the pointer to the QALoad ADO Connect object for this instance of the actual ADO Command object.

Note: The n associated with the ADO Command object and the n associated with the ADOConnect parameter reference different instances of different QALoad ADO replay objects.

Syntax

```
ADO_Command(n)->PutRefActiveConnection( CAConnect* pADOConnect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pADOConnect	The connection object (ADO Connect).

Example

```
ADO_Command(1)->PutRefActiveConnection( ADOConnect[0] );
ADO_Command(1)->PutCommandType( adCmdStoredProc );
ADO_Command(1)->PutCommandText( "op_Getparamvb6_Batch" );
BeginCheckpoint("ADOCommand::GetParameters");
ADO_Command(1)->GetParameters( ADOParameterSet[0] );
EndCheckpoint("ADOCommand::GetParameters");
```

ADO_Connect(n)->BeginTrans

Begins a new transaction.

If your provider supports nested transactions, you can call the BeginTrans method within an open transaction to start a new, nested transaction. The method returns the level of nesting: 1 indicates a top-level transaction, 2 indicates a second-level transaction, and so forth.

Syntax

```
ADO_Connect(n)->BeginTrans( long* pTransactionLevel );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pTransactionLevel	Pointer to a long containing the level of nesting used on this transaction.

Example

```
ADO_Connect(1)->BeginTrans( pLong );
BeginCheckpoint("ADOConnect::Execute");
ADO_Connect(1)->Execute( "DELETE FROM Temp WHERE HI =" 291667 ", pvValue, -1,
ADORecordset[7] );
EndCheckpoint("ADOConnect::Execute");
ADORecordset.Release( 7, ADOBM );
```

ADO_Connect(n)->Close

Closes a Connection object.

This method closes a Connection object and any active Recordset objects associated with the Connection. Any Command object associated with the Connection object still exists, but is no longer associated with a Connection object.

Note: There is currently no provision to independently track the Recordsets or command associated with each different Connection object.

Syntax

```
ADO_Connect(n)->Close();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Connect( 1 )->Close( );
ADOConnect.Release( 1 );
```

ADO_Connect(n)->CommitTrans

Save changes, ends the transaction. May start a new transaction.

Only affects the most recently opened transaction. Close or rollback a transaction to resolve higher-level transactions.

After you call the BeginTrans method, the provider no longer instantaneously commits changes you make until you call CommitTrans or RollbackTrans to end the transaction.

Syntax

```
ADO_Connect(n)->CommitTrans();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Connect(1)->BeginTrans ( pLong );
ADO_Recordset(18)->GetState ( pLong );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(FIRSTNAME)" );
ADO_LoadVariant( pvValue, "","","" );
ADO_Recordset(15)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(LASTNAME)" );
ADO_LoadVariant( pvValue, "", "","" );
ADO_Recordset(11)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(ADDRESS)" );
ADO_LoadVariant( pvValue, "","","" );
ADO_Recordset(12)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_Recordset(27)->Close();
ADORecordset.Release( 27, ADOBM );
ADO_LoadVariant( pvSource, "8", "select max(lsystemID) from CITY" );'
LoadVariant( pvValue, ADOConnect[1[ );
ADO_Recordset(13)->Open( pvSource, pvValue, adOpenForwardOnly, adLockReadOnly, -1 );
ADORecordset.Release( 13, ADOBM );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(19)->Update( pvValue, pvData );
ADO_Connect(1)->CommitTrans();
```

ADO_Connect(n)->Execute

Executes the query passed in the CommandText argument on the connection to the method.

If a row-returning query is specified, results are stored in a new ADO Recordset object. If a row-returning query is not specified, a closed ADO Recordset object is returned.

Syntax

```
ADO_Connect->Execute( char* sCommand, VARIANT* pvRecsAffected, long options, CARecordSet*
pADORecordSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sCommand	An ANSI representation of the command string.
pvRecsAffected	A pointer to the variant returning the number of records affected by this call.
options	The options used to make this call run.
pADORecordSet	The *QA*Load object containing the new recordset created by this call. The Recordset that is returned is returned inside of the identified ADORecordset(#) object.

Example

```
BeginCheckpoint("ADOConnect::Execute");
ADO_Connect->Execute( "Command", pvValue, #, ADORecordset(#) );
EndCheckpoint("ADOConnect::Execute");
ADORecordset.Release( 0, ADOBM );
```

ADO_Connect(n)->GetAttributes

GetAttributes is read/write. It's value is the sum of one or more XactAttributeEnum values. The default is zero (0).

Syntax

```
ADO_Connect(n)->GetAttributes( long* pAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pAttributes	A pointer to a long.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 262144 );
ADO_Connect(0)->PutAttributes( 393216 );
ADO_Connect(0)->GetAttributes( pLong );
```

ADO_Connect(n)->GetCommandTimeout

Returns the value of the timeout in a pointer to a long.

Use GetCommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled for network or server delays. If the command does not excute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Connect(n)->GetCommandTimeout( long* pCommandTimeout );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCommandTimeout	A pointer to a long.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->GetConnectionString

GetConnectionString method retrieves the value of ConnectionString property of this instance of the ADO Connect object.

The ConnectionString specifies a data source by passing argument=value statements. These are separated by semi-colons.

Syntax

```
ADO_Connect(n)->GetConnectionString( CLoadString& sConnectionString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sConnectionString	This CLoadString retrieves the value of the ConnectionString property for this instance of ADO Connection.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 0 );
ADO_Connect(0)->GetConnectionString( sLoadStr );
ADO_Connect(0)->PutConnectionString( "DSN=LoadTestBox;UID=SA;PWRD=H" );
```

ADO_Connect(n)->GetConnectionTimeout

Use GetConnectionTimeout on a Connection object to cancel a connection attempt, if necessary, due to network or server delays.

If the time interval specified in the Connection Timeout property setting runs out before a connection can be opened, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

Note: Before using GetConnection Timeout, ensure that your provider supports ADO's ConnectionTimeout functionality.

Syntax

```
ADO_Connect(n)->GetConnectionTimeout( long* pConnectTimeout );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pConnectTimeout	A pointer to a long returning the value in seconds of the connection timeout property of this ADOConnection object.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->GetCursorLocation

Enables selection of a cursor library from those accessible to the provider.

You can usually choose to use a client-side or server-side location. This setting only affects those connections made after setting the property. It has no effect on existing connections.

Syntax

```
ADO_Connect(n)->GetCursorLocation( long* pCursorLocation );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCursorLocation	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Connect(0)->GetCursorLocation( pLong );
ADO_Connect(0)->PutCursorLocation( adUseServer );
ADO_Connect(0)->GetCommandTimeout( pLong );
```

ADO_Connect(n)->GetDefaultDatabase

Retrieves the value of the DefaultDatabase property from this instance of the ADO Connect object.

Syntax

```
ADO_Connect(n)->GetDefaultDatabase( CLoadString& sDBString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sDBString	The CLoadString that returns the DefaultDatabase information to the script.

Example

```
ADO_Connect(0)->Open("PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa; pwd="";database=Master;", "",
"", -1 );
ADO_Connect(0)->GetDefaultDatabase( sLoadStr );
ADO_Connect(0)->PutDefaultDatabase("pubs" );
```

ADO_Connect(n)-> GetIsolationLevel

Sets a Connection object's isolation level. Takes effect the next time the BeginTrans method is called.

Syntax

```
ADO_Connect(n)-> GetIsolationLevel( pLong );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
long* pIsolationLevel	A pointer to a long returning the value of the isolation level property of this instance of the ADO Connection object.

Example

```
ADO_Connect(0)->GetIsolationLevel( pLong );
ADO_Connect(0)->PutIsolationLevel( adXactSerializable );
```

ADO_Connect(n)->GetMode

Sets or returns access permissions for the current connection.

The GetMode property can only be set after the Connection object is closed.

Syntax

```
ADO_Connect(n)->GetMode( long* pMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pMode	Retrieves the ConnectionModeEnum from the call and converts that to a long* to be returned to the script.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );
ADO_Connect(0)->GetProvider( sLoadStr6 );
```

ADO_Connect(n)->GetProvider

Returns the provider name for a connection.

You can also set this property using the contents of the Open method's ConnectionString property or argument. Note that you may get unwanted results if you specify a provider in more than one place while using the Open method. If a provider is not specified, this property defaults to MSDASQL.

When the connection is closed, GetProvider is read/write. When the connection is open, GetProvider is read-only. Before the setting can take effect, you must open the Connection object or access the Connection object's Properties collection. An invalid setting results in an error.

Syntax

```
ADO_Connect(n)->GetProvider( CLoadString& sProvider );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sProvider	This CLoadString holds the Provider information returned by this call.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );
ADO_Connect(0)->GetProvider( sLoadStr );
```

ADO_Connect(n)->GetState

Determines the state of a specified object at any time.

This property can have a combination of values.

Syntax

```
ADO_Connect(n)->GetState( long* pState );
```

Return Value

0 for closed
1 for open

Parameters

Parameter	Description
n	An index to the object.
pState	A pointer to a long holding the state information (0 or 1).

Example

```
ADO_Connect(0)->PutProvider( "MSDASQL" );
ADO_Connect(0)->GetState( pLong );
ADO_Connect(0)->GetVersion( sLoadStr );
```

ADO_Connect(n)->GetVersion

Returns the version of ADO that is being used.

The version is available as a dynamic property in the Properties collection.

Syntax

```
ADO_Connect(n)->GetVersion( CLoadString& sVersion );
```

Return Value

Parameters

Parameter	Description
n	An index to the pointer.
sVersion	The CLoadString string that returns version to the script.

Example

```
ADO_Connect(0)->PutProvider( "MSDASQL" );
ADO_Connect(0)->GetState( pLong );
ADO_Connect(0)->GetVersion( sLoadStr );
```

ADO_Connect(n)->Open

Establishes the connection to the data source.

When successful, it creates a live connection against which you can issue commands.

Syntax

```
ADO_Connect(n)->Open( char* sConnectString, char* sUser, char* sPassword,
ADOConnectOptionEnum option );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sConnectString	The ConnectionString property automatically inherits the value used for the ConnectionString argument. Therefore, you can either set the ConnectionString property of the Connection object before opening it, or use the ConnectionString argument to set or override the current connection parameters during the Open method call.

sUser	The UserID and Password arguments will override the values specified in ConnectionString.
sPassword	See the User parameter.
option	ADOConnectOptionEnum A ConnectOptionEnum value that determines whether this method should return after (synchronously) or before (asynchronously) the connection is established. Valid values are: Value — Description adConnectUnspecified — Unspecified connection option adAsyncConnect — Asynchronous connect option

Example

```
ADO_Connect(1)->PutConnectionString( "DSN=My;UID=sa; PWD="";" );
BeginCheckpoint("ADOConnect::Open");
ADO_Connect(1)->Open( "", "", "", -1 );
EndCheckpoint("ADOConnect::Open");
ADO_Connect(1)->PutCommandTimeout( 200 );
```

ADO_Connect(n)->OpenSchema

Returns information about the data source. For example, tables, columns included in the tables, and data types.

Syntax

```
ADO_Connect(n)->OpenSchema( ADOSchemaEnum schema, VARIANT* pvRestrictions, VARIANT*
pvSchemaID, CARecordSet* pADORecordset );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
schema	ADOSchemaEnum SchemaEnum describing the type of schema query to run. Valid values are: Value — Description adSchemaAsserts — Returns the assertions defined in the catalog that are owned by a given user adSchemaCatalogs — Returns the physical attributes associated with catalogs accessible from the DBMS

	adSchemaCharacterSets	Returns the character sets defined in the catalog that are accessible to a given user
	adSchemaCheckConstraints	Returns the check constraints defined in the catalog that are owned by a given user
	adSchemaCollations	Returns the character collations defined in the catalog that are accessible to a given user
	adSchemaColumnPrivileges	Returns the privileges on columns of tables defined in the catalog that are available to, or granted by, a given user
	adSchemaColumns	Returns the columns of tables (including views) defined in the catalog that are accessible to a given user
	adSchemaColumnsDomainUsage	Returns the columns defined in the catalog that are dependent on a domain defined in the catalog and owned by a given user
	adSchemaConstraintColumnUsage	Returns the columns used by referential constraints, unique constraints, check constraints, and assertions, defined in the catalog and owned by a given user
	adSchemaConstraintTableUsage	Returns the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in the catalog and owned by a given user
	adSchemaCubes	Returns information about the available cubes in a schema (or the catalog, if the provider does not support schemas)
	adSchemaDBInfoKeywords	Returns a list of provider-specific keywords
	adSchemaDBInfoLiterals	Returns a list of provider-specific literals used in text commands
	adSchemaDimensions	Returns information about the dimensions in a given cube. It has one row for each dimension
	adSchemaForeignKeys	Returns the foreign key columns defined in the catalog by a given user
	adSchemaHierarchies	Returns information about the hierarchies available in a dimension
	adSchemaIndexes	Returns the indexes defined in the catalog that are owned by a given user
	adSchemaKeyColumnUsage	Returns the columns defined in the

		catalog that are constrained as keys by a given user
	adSchemaLevels	Returns information about the levels available in a dimension
	adSchemaMeasures	Returns information about the available measures
	adSchemaMembers	Returns information about the available members
	adSchemaPrimaryKeys	Returns the primary key columns defined in the catalog by a given user
	adSchemaProcedureColumns	Returns information about the columns of rowsets returned by procedures
	adSchemaProcedureParameters	Returns information about the parameters and return codes of procedures
	adSchemaProcedures	Returns the procedures defined in the catalog that are owned by a given user
	adSchemaProperties	Returns information about the available properties for each level of the dimension
	adSchemaProviderSpecific	Used if the provider defines its own nonstandard schema queries
	adSchemaProviderTypes	Returns the (base) data types supported by the data provider
	adSchemaReferentialConstraints	Returns the referential constraints defined in the catalog that are owned by a given user
	adSchemaSchemata	Returns the schemas (database objects) that are owned by a given user
	adSchemaSQLLanguages	Returns the conformance levels, options, and dialects supported by the SQL-implementation processing data defined in the catalog
	adSchemaStatistics	Returns the statistics defined in the catalog that are owned by a given user
	adSchemaTableConstraints	Returns the table constraints defined in the catalog that are owned by a given user
	adSchemaTablePrivileges	Returns the privileges on tables defined in the catalog that are available to, or granted by, a given user
	adSchemaTables	Returns the tables (including views) defined in the catalog that are accessible

		to a given user
	adSchemaTranslations	Returns the character translations defined in the catalog that are accessible to a given user
	adSchemaTrustees	Reserved for future use
	adSchemaUsagePrivileges	Returns the USAGE privileges on objects defined in the catalog that are available to, or granted by, a given user
	adSchemaViewColumnUsage	Returns the columns on which viewed tables, defined in the catalog and owned by a given user, are dependent
	adSchemaViews	Returns the views defined in the catalog that are accessible to a given user
	adSchemaViewTableUsage	Returns the tables on which viewed tables, defined in the catalog and owned by a given user, are dependent
pvRestrictions	An array of query constraints for each QueryType option, as listed in SchemaEnum.	
pvSchemaID	This parameter is required if QueryType is set to adSchemaProviderSpecific; otherwise, it is not used.	
pADORecordset	The recordset composing the result set.	

Note: The Accompanying ADO_LoadVariant calls are made to set the proper values for the VARIANTs pvValue and pvData. They are included automatically by QALoad 's conversion process.

ADO_Connect(n)->PutAttributes

Sets the transaction attribute for this connection object.

The number represented by the string in the call is used by ADO during the call.

Syntax

```
ADO_Connect(n)->PutAttributes( long attribute );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
attribute	The sum of one or more XactAttributeEnum values. The default is 0. This property is read/write.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes((XactAttributeEnum) 262144 );
ADO_Connect(0)->GetAttributes( pLong );
```

ADO_Connect(n)->PutCommandTimeout

PutCommandTimeout Sends a timeout in seconds before the command will timeout with an error.

Use PutCommandTimeout on a Connection object or Command object to allow an Execute method call to be cancelled due to network or server delays. If the command does not execute before the time interval runs out, an error occurs and the command is cancelled. Set the property to zero to wait indefinitely.

Syntax

```
ADO_Connect(n)->PutCommandTimeout( long seconds );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
seconds	The number of seconds before an exception is generated by the command overrunning the timeout.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->PutConnectionString

Specifies a data source.

PutConnectionString passes detailed connection strings that include argument=value statements. Use semi-colons to separate the argument=value statements.

Syntax

```
ADO_Connect(n)->PutConnectionString( CLoadString& sConnectionString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sConnectionString	This ANSI string sets the value of the ConnectionString property for this instance of ADO Connection.

Example

```
ADO_Connect(0)->GetAttributes( pLong );
ADO_Connect(0)->PutAttributes( 0 );
ADO_Connect(0)->GetConnectionString( sLoadStr );
ADO_Connect(0)->PutConnectionString( "DSN=LoadTestBox;UID=SA;PWRD=H" );
```

ADO_Connect(n)->PutConnectionTimeout

Use PutConnectionTimeout on a Connection object to abandon an attempt to connect due to network or server delays.

If a connection is not made in the specified time, an error occurs and the attempt to connect is cancelled. Set the property to zero to wait indefinitely.

Note: Before using PutConnection Timeout, ensure that your provider supports ADO's ConnectionTimeout functionality.

Syntax

```
ADO_Connect(n)->PutConnectionTimeout( long seconds );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
seconds	The number of seconds to attempt a connection to the provider.

Example

```
ADO_Connect(0)->PutCommandTimeout( 15 );
ADO_Connect(0)->GetCommandTimeout( pLong );
ADO_Connect(0)->GetConnectionTimeout( pLong );
ADO_Connect(0)->PutConnectionTimeout( 250 );
```

ADO_Connect(n)->PutCursorLocation

Lets you choose a cursor library from those accessible to the provider.

You can usually choose to use a client-side or server-side library. This setting only affects those connections made after setting the property. It has no effect on existing connections.

Syntax

```
ADO_Connect(n)->PutCursorLocation( ADOCursorLocationEnum nCursorLoc );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nCursorLoc	ADOCursorLocationEnum A String representation of a CursorLocationEnum value. Valid values are: Value — Description adUseNone — No specified cursor location adUseServer — Server-side cursor adUseClient — Client-side cursor adUseClientBatch — Batch client-side cursor

Example

```
ADO_Connect(0)->GetCursorLocation( pLong );
ADO_Connect(0)->PutCursorLocation( adUseServer );
ADO_Connect(0)->GetCommandTimeout( pLong );
```

ADO_Connect(n)->PutDefaultDatabase

Sets the default database within a connection object.

Access objects in a database other than the one specified in the DefaultDatabase property by qualifying object names with the desired database name. Upon connection, the provider writes default database information to the DefaultDatabase property.

Syntax

```
ADO_Connect(n)->PutDefaultDatabase( char* sDBString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

sDBString	The name of the default database to set within the connection object.

Example

```
ADO_Connect(0)->Open("PROVIDER=MSDASQL;dsn=FhLoadDB2;uid=sa;
                     pwd="";database=Master;", "", "", -1 );
ADO_Connect(0)->GetDefaultDatabase( sLoadStr );
ADO_Connect(0)->PutDefaultDatabase("pubs" );
```

ADO_Connect(n)->PutIsolationLevel

Sets the isolation level of a Connection object. The isolation level takes effect the next time you call BeginTrans.

The isolation level that is part of this call is represented as a string. This string representation translates into a number that is used by ADO internally.

Syntax

```
ADO_Connect(n)->PutIsolationLevel( ADOIsolationLevelEnum nIsoLevel );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nIsoLevel	ADOIsolationLevelEnum The isolation level, either in numeric format or in the adXactSerializable format. Possible values are: Value — Description adXactUnspecified — Unspecified isolation level adXactChaos — No isolation level adXactReadUncommitted — Read uncommitted isolation level adXactBrowse — Browse isolation level adXactCursorStability — Cursor stability isolation level adXactReadCommitted — Read committed isolation level adXactRepeatableRead — Repeatable read isolation level adXactSerializable — Serializable isolation level adXactIsolated — Isolated

Example

```
ADO_Connect(0)->GetIsolationLevel( pLong );
ADO_Connect(0)->PutIsolationLevel( adXactSerializable );
```

ADO_Connect(n)->PutMode

Sets the access permissions being used on the current connection.

Note: This property can only be set when the Connection object is closed.

Syntax

```
ADO_Connect(n)->PutMode( ADOConnectModeEnum nConnectMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nConnectMode	ADOConnectModeEnum Sets the ConnectModeEnum property for this instance of the ADO Connect object. Valid values are: Value — Description adModeUnknown — Unknown connection mode adModeRead — Read-only mode adModeWrite — Write-only mode adModeReadWrite — Read-write mode adModeShareDenyRead — Exclusive read mode adModeShareDenyWrite — Exclusive write mode adModeShareExclusive — Exclusive read-write mode adModeShareDenyNone — Non-exclusive mode adModeRecursive — Recursive mode

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );
ADO_Connect(0)->GetProvider( sLoadStr6 );
```

ADO_Connect(n)->PutProvider

Sets the provider name for a connection.

You can also set this property using the contents of the Open method's ConnectionString property or argument. Note that you may get unwanted results if you specify a provider in more than one place while using the Open method. If a provider is not specified, this property defaults to MSDASQL.

When the connection is closed, PutProvider is read/write. When the connection is open, PutProvider is read only. Before the setting can take effect, you must open the Connection object or access the Connection object's Properties collection. An invalid setting results in an error.

Syntax

```
ADO_Connect(n)->PutProvider( char* sProviderString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sProviderString	The name of the provider being used to access data.

Example

```
ADO_Connect(0)->PutIsolationLevel( adXactReadCommitted );
ADO_Connect(0)->GetMode( pLong );
ADO_Connect(0)->PutMode( (ConnectModeEnum)12 );
ADO_Connect(0)->PutProvider( "MSDASQL" );
```

ADO_Connect(n)->RollbackTrans

Reverses changes made in an open transaction and ends the transaction.

This is linked with BeginTrans. This is seen only in the script if a transaction fails for some reason. If it fails and you see this call, look over the script logic and see if the transaction can be committed.

Syntax

```
ADO_Connect(n)->RollbackTrans();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Connect(1)->BeginTrans( pLong );
ADO_Recordset(18)->GetState( pLong );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(FIRSTNAME)" );
ADO_LoadVariant( pvValue, "", "","" );
ADO_Recordset(15)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(LASTNAME)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(11)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_LoadVariant( pvSource, "8", "SELECT rtrim(ADDRESS)" );
ADO_LoadVariant( pvValue, "", "", "" );
ADO_Recordset(12)->Open( pvSource, pvValue, adOpenForwardOnly, 0, 0 );
ADO_Recordset(27)->Close();
ADORecordset.Release( 27, ADOBM );
ADO_LoadVariant( pvSource, "8", "select max(lsystemID) from CITY" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(13)->Open( pvSource, pvValue, adOpenForwardOnly, adLockReadOnly, -1 );
ADORecordset.Release( 13, ADOBM );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(19)->Update( pvValue, pvData );
ADO)Connect(1)->Rollback();
```

ADO_Field(n)->AppendChunk

Special data handling method that writes data in chunks to the Field object.

This is especially useful when memory is limited, since you can use this method to manipulate long values in manageable chunks. It may take numerous calls to AppendChunk to completely write the data to the appropriate field object. When writing data values using ADO, the datatype being used as the parameter with the data value is often a VARIANT datatype.

The first call to AppendChunk writes data to the field and overwrites any existing data. Subsequent calls add to the data. Note that if you append data to one field, then manipulate another field in the same record, ADO assumes you are finished with the first field. If you then attempt to append data to the first field, the existing data is overwritten.

The AppendChunk call is preceded immediately in the script by a call to ADO_LoadVariant.

You can use the AppendChunk method in the Attributes property of a Field object if the adFldLong bit in the Attributes property is set to true.

Syntax

```
ADO_Field(n)->AppendChunk( VARIANT* pvValue )
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	This pointer to a variant contains a chunk of data to be written to the field object.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SUSERRESUME" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Roger Smith\n2114 Edgemere Court\nGrosse Pointe" ",
                MI 48263\n\n\nObjective:\n \nA job controlling the Universe"
                "which I can do from the comfort of my own home." );
ADO_Field(0)->AppendChunk( pvValue );
ADO_LoadVariant( pvValue, "8", "\n\nExperience:\nCEO for General Motors during"
                "the Reagan administration. CEO for General Electric during"
                "the Carter Administration .. \n" );
ADO_Field(0)->AppendChunk( pvValue ); /* Type: 8 - VT_BSTR Data: admin */
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724");
ADO_Recordset(18)->Update( pvValue, pvData );
```

ADO_Field(n)->GetActualSize

Retrieves the value contained within the ActualSize property of this instance of the ADO Field object.

GetActualSize returns a Long value. If your provider allows this property to reserve space for BLOB data, the default is zero.

GetActualSize is read-only. If ADO fails to identify the length of the object's value, this property returns adUnknown.

Syntax

```
ADO_Field(n)->GetActualSize( long* pActualSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pActualSize	A pointer to a long, which contains the value of the ActualSize property of this instance of the ADO Field Object.

Example

```
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
ADO_Field(3)->GetActualSize( pLong );
```

ADO_Field(n)->GetAttributes

Retrieves the value contained within the Attributes property of this instance of the ADO Field object.

The Attributes property can be the sum of one or more FieldAttributeEnum values, and it is normally read-only. It is read/write if all of the following conditions are met:

- it is a new Field object
- it has been appended to the Fields collection of a Record
- the Value property for the Field has been specified
- the new Field has been added by the data provider (using the Fields collection's Update method)

Syntax

```
ADO_Field(n)->GetAttributes( long* pAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pAttributes	A pointer to a long, which contains the value of the Attributes property of this instance of the ADO Field Object.

Example

```
ADO_Field(1)->PutAttributes( 32 );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 100 );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->GetChunk

Retrieves chunks of binary or character data to an appropriate buffer.

This is a special function that may have to be called several times in order to properly handle the data being returned from the field object. Used on a Field object, this method returns all or part of the object's long binary or character data. Use this method to manipulate long values in manageable chunks of data.

Since the data is being returned from this call, the ADO_LoadVariant is not called. The returned data is assigned to a variable. If the variable size is greater than the remaining data, only the data is returned. The variable is not padded with empty spaces. An empty field returns a null value.

If more than one call to GetChunk is necessary, each subsequent call starts retrieving data from the point where the previous call stopped. Note that if you retrieve data from one field and then manipulate another field, ADO assumes you are finished with the first field.

Syntax

```
ADO_Field(n)->GetChunk( long nLength, VARIANT* pDataReturned );
```

Return Value

Parameters

Parameter	Description

n	An index to the object.
nLength	Length of the chunk of data being retrieved.
pDataReturned	Pointer to the variant holding the chunk of retrieved data.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SUSERRESUME" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetChunk( 100, pvValue );
ADO_Field(0)->GetChunk( 100, pvValue );
```

ADO_Field(n)->GetDataFormat

Retrieves an IUnknown instance describing the data format for this field.

Syntax

```
ADO_Field(n)->GetDataFormat( &pIUnknown pDataFormat );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pDataFormat	Pointer to an IUnknown interface.

Example

```
ADO_Field(1)->GetDataFormat( &pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->GetDefinedSize

Retrieves the value contained within the DefinedSize property of this instance of the ADO Field object.

Used to determine the data capacity of a Field object.

Syntax

```
ADO_Field(n)->GetDefinedSize( long* pDefinedSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pDefinedSize	A pointer to a long with the value of the DefinedSize property of this Instance of the ADO Field object.

Example

```
ADO_Field(1)->GetDataFormat( pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 4 );
```

ADO_Field(n)->GetName

Retrieves the value contained within the Name property of this instance of the ADO Field object.

Note that the actual ADO call has a BSTR as the argument, so there is some data conversion occurring within this call. GetName is normally read-only. It is read/write if all of the following conditions are met:

- It is a new Field object
- The new Field object has been appended to the Fields collection of a Record
- The Value property for the Field has been specified
- The data provider has added the new Field (using the Fields collection's Update method)

Syntax

```
ADO_Field(n)->GetName( CLoadString& sName );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sName	Value of the Name property for this ADO Field Object.

Example

```
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
ADO_Field(3)->GetActualSize( pLong );
```

ADO_Field(n)->GetNumericScale

Retrieves the value contained within the NumericScale property of this instance of the ADO Field object.

GetNumericScale sets or returns a byte value indicating the number of decimal places to which numeric values are resolved. GetNumericScale is normally read-only. It is read/write if all of the following conditions are met:

- it is on a new Field object
- the new Field object has been appended to the Fields collection of a Record
- the Value property for the Field has been specified
- the data provider has added the new Field (using the Fields collection's Update method)

Syntax

```
ADO_Field(n)->GetNumericScale( unsigned char* sChar );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sChar	Pointer to an unsigned character

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( &cUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->GetOriginalValue

Retrieves the value contained within the Value property of this instance of the ADO Field object.

This occurs before any changes are made permanent by a call to an update method.

Syntax

```
ADO_Field(n)->GetOriginalValue( VARIANT* pOriginalValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pOriginalValue	The pointer to the VARIANT in which the original value is returned.

Example

```
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetOriginalValue( pvValue );
```

ADO_Field(n)->GetPrecision

Retrieves the value contained within the Precision property of this instance of the ADO Field object.

You can use it to determine the maximum number of digits used to represent a numeric Parameter or field object. Note that the data being returned is in the form of a pointer to an unsigned char. This is essentially a byte depicting the size of a column from 0 bytes to 256 bytes.

Syntax

```
ADO_Field(n)->GetPrecision( unsigned char* sChar );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sChar	Pointer to an unsigned character

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( &cUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->GetProperties

Retrieves the complete set of properties for this particular instance of the Field object.

PropertySets may change for different providers.

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider.

Syntax

```
ADO_Field(n)->GetProperties( CAPropertySet* pPropertySet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pPropertySet	Set of CAProperty objects. Each CAProperty object contains a single characteristic, a piece of data, which partially describes the state of a particular instance of an object.

Example

```
ADO_LoadVariant( pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Field(n)->GetStatus

Retrieves the value contained within the Status property of this instance of the ADO Field object.

It takes a pointer to a long as an argument. Within this parameter, the value returned is the value of the status property of this instance of the Field object.

Syntax

```
ADO_Field(n)->GetStatus( long* pStatus );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pStatus	Pointer to the value within the Status property of this instance of the ADO Field object.

Example

```
ADO_Field(2)->GetStatus( pLong );
```

ADO_Field(n)->GetType

Retrieves the value contained within the Type property of this instance of the ADO Field object.

The Actual ADO call uses another enumerated type, DataTypeEnum, to handle the data types. Conversion to this type happens within the QALoad call.

GetType is read/write if all of the following conditions are met:

- it ion a new Field object
- the new Field object has been appended to the Fields collection of a Record
- the Value property for the Field has been specified
- the data provider has added the new Field (using the Fields collection's Update method)

Syntax

```
ADO_Field(n)->GetType( long* pType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pType	A pointer to a long containing the data type of the field element.

Example

```
ADO_Field(2)->PutPrecision( 0x00 );
ADO_Field(1)->GetType( pLong );
ADO_Field(1)->PutType( (DataTypeEnum)8 );
```

ADO_Field(n)->GetUnderlyingValue

Specifies the current value of a Field object in the database after any updates to the recordset.

This is the current value visible to your transaction. It may be the result of a recent update by another transaction. Note that this could be different from the OriginalValue property that was originally returned to the Recordset.

Syntax

```
ADO_Field(n)->GetUnderlyingValue( VARIANT* pUnderlyingValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pUnderlyingValue	The pointer to the variant holding the underlying value returned from the call.

Example

```
ADO_Field(3)->PutValue( pvValue );
ADO_Field(3)->GetUnderlyingValue( pvValue );
ADO_Field(3)->GetName( sLoadStr );
```

ADO_Field(n)->GetValue

Retrieves the value of an ADO Field object into a pointer to a Variant.

The argument handles any type of data by using the Variant datatype. Data is retrieved into a pointer to a Variant.

Syntax

```
ADO_Field(n)->GetValue( VARIANT* pValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pValue	Pointer to the value contained within the Value property of this instance of the ADO Field object.

Example

```
ADO_Field(3)->GetValue( pvValue );
LoadVariant( pvValue, "8", "T " );
ADO_Field(3)->PutValue( pvValue );
```

ADO_Field(n)->PutAttributes

The PutAttributes method call sets the value contained within the Attributes property of this instance of the ADO Field object.

This property can be the sum of one or more FieldAttributeEnum values. It is normally read-only; however, it can be read/write if all of the following conditions are present:

- it is a new Field object
- the new Field object has been appended to the Fields collection of a Record
- the Value property for the Field has been specified
- the new Field has been successfully added by the data provider (using the Field collection's Update method).

Syntax

```
ADO_Field(n)-PutAttributes ( ADOFieldAttributeEnum nAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAttributes	ADOFieldAttributeEnum The field attribute. Valid values are:

	Value	Description
	adFldUnspecified	Unspecified attribute
	adFldMayDefer	MayDefer attribute
	adFldUpdatable	Updatable attribute
	adFldUnknownUpdatable	Unknown Updatable attribute
	adFldFixed	FldFixed attribute
	adFldIsNullable	IsNullable attribute
	adFldMayBeNull	FldMayBeNull attribute
	adFldLong	FldLong attribute
	adFldRowID	FldRowID attribute
	adFldRowVersion	RowVersion attribute
	adFldCacheDeferred	FldCacheDeferred attribute
	adFldIsChapter	FldIsChapter attribute
	adFldNegativeScale	FldNegativeScale attribute
	adFldIsRowURL	FldIsRowURL attribute
	adFldIsDefaultStream	FldIsDefaultStream attribute
	adFldIsCollection	FldIs Collection attribute

Example

```
ADO_Field(1)->PutAttributes((FieldAttributeEnum)32 );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 100 );
ADO_Field(1)->GetDefinedSize( pLong );
```

ADO_Field(n)->PutDefinedSize

Sets the value contained within the DefinedSize property of this instance of the ADO Field object.

Note that the DefinedSize and ActualSize properties are different. For example, if a DefinedSize property of 25 only contained a few characters, the ActualSize property value would be the length of those few characters.

Syntax

```
ADO_Field(n)->PutDefinedSize( long nSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nSize	Size of the data contained within this field.

Example

```
ADO_Field(1)->GetDataFormat( pIUnknown );
ADO_Field(1)->GetAttributes( pLong );
ADO_Field(1)->GetDefinedSize( pLong );
ADO_Field(1)->PutDefinedSize( 4 );
```

ADO_Field(n)->PutNumericScale

Sets the value contained within the NumericScale property of this instance of the ADO Field object.

PutNumericScale also sets or returns a byte value indicating the number of decimal places to which numeric values are resolved. PutNumericScale is normally read-only; however, it is read/write if all of the following conditions are met:

- it applies to a new Field object
- the new Field object has been appended to the Fields collection of a Record
- the Value property for the Field has been specified
- the data provider has added the new Field (using the Fields collection's Update method)

Syntax

```
ADO_Field(n)->PutNumericScale( unsigned char ucNumericScale );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
ucNumericScale	Unsigned character being passed in its hexadecimal representation. Format: 0x## (Note that the range on the # is 0 - F - 0123456789ABCDEF)

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( pUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->PutPrecision

Sets the value contained within the Precision property of this instance of the ADO Field object.

You can use this to determine the maximum number of digits used to represent a numeric Parameter or Field object.

Syntax

```
ADO_Field(n)->PutPrecision( unsigned char ucPrecision );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
ucPrecision	Unsigned character being passed in its hexadecimal representation. Format: 0x## (Note that the range on the # is 0 - F - 0123456789ABCDEF)

Example

```
ADO_Field(1)->PutDefinedSize( 4 );
ADO_Field(2)->GetNumericScale( pUChar );
ADO_Field(2)->PutNumericScale( 0x03 );
ADO_Field(2)->GetPrecision( pUChar );
ADO_Field(2)->PutPrecision( 0x07 );
```

ADO_Field(n)->PutRefDataFormat

This updates the current value of the data format updates to the ADO Recordset.

Syntax

```
ADO_Field(n)->PutRefDataFormat( IUnknown* pIUnknown );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pIUnknown	Pointer to the IUnknown interface.

ADO_Field(n)->PutType

Sets the value contained within the Type property of this instance of the ADO Field object.

There is a conversion from the long argument to a DataTypeEnum*, which is accomplished through a cast. PutType is read/write for a new Field object that has been appended to a Record's Fields collection if the following conditions are met:

- the Value property for the Field has been specified
- the data provider has added the new Field (using the Fields collection's Update method).

Syntax

```
ADO_Field(n)->PutType( ADODataTypeEnum nType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nType	ADODataTypeEnum The datatype of the field element. Valid values are: Value — Description adEmpty — No data type specified adTinyInt — Tiny integer adSmallInt — Small integer adInteger — Integer adBigInt — Big integer adUnsignedTinyInt — Unsigned tiny integer adUnsignedSmallInt — Unsigned small integer adUnsignedInt — Unsigned integer adUnsignedBigInt — Unsigned integer adSingle — Single precision float adDouble — Double precision float adCurrency — Currency adDecimal — Decimal adNumeric — Numeric adBoolean — Boolean

	adError	Error
	adUserDefined	User-defined data type
	adVariant	Variant
	adIDispatch	Pointer to IDispatch
	adIUnknown	Pointer to IUnknown
	adGUID	GUID
	adDate	Date
	adDBDate	DBDate
	adDBTime	DBTime
	adDBTimeStamp	DBTimestamp
	adBSTR	BSTR
	adChar	Char
	adVarChar	VarChar
	adLongVarChar	Long VarChar
	adWChar	Wide Char
	adVarWChar	VarWChar
	adLongVarWChar	Long VarWChar
	adBinary	Binary
	adVarBinary	VarBinary
	adLongVarBinary	Long VarBinary
	adChapter	Chapter
	adFileTime	FileTime
	adPropVariant	Variant Property
	adVarNumeric	VarNumeric
	adArray	Array

Example

```
ADO_Field(2)->PutPrecision( 0x00 );
ADO_Field(1)->GetType( pLong );
ADO_Field(1)->PutType( (DataTypeEnum)8 );
```

ADO_Field(n)->PutValue

Resets the value of this instance of the ADO Field object. This is the first step in updating a Recordset's value.

In order to accommodate any type of data, ADO uses the Variant datatype with PutValue.

Note: This call, and all other calls that use the Variant datatypes, have to use an accompanying Setup function call, the call to ADO_LoadVariant.

Syntax

```
ADO_Field(n)->PutValue( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the variant that returns the data contained in the ADO Field object.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "active" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Y" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" ); // VT_ERROR;
ADO_LoadVariant( pvData, "10", "2147614724" ); // VT_ERROR;
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
ADORecordset.Release( 2, ADOBM );
```

ADO_FieldSet(n)->Append

Creates and appends a new Field object to the ADO FieldSet.

An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprises a mechanism for updating or retrieving information from a Data Provider.

Syntax

```
ADO_FieldSet(n)->Append( char* sNameString, ADODataTypeEnum nDT, long nDefinedSize,
ADOFieldAttributeEnum nFA, VARIANT* pvFieldValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sNameString	The name of the Field being added to the collection of fields.
nDT	ADODataTypeEnum The Data type of the field being added to the collection. Valid values are: Value — Description adEmpty — No data type specified adTinyInt — Tiny integer adSmallInt — Small integer adInteger — Integer adBigInt — Big integer adUnsignedTinyInt — Unsigned tiny integer adUnsignedSmallInt — Unsigned small integer adUnsignedInt — Unsigned integer adUnsignedBigInt — Unsigned integer adSingle — Single precision float adDouble — Double precision float adCurrency — Currency adDecimal — Decimal adNumeric — Numeric adBoolean — Boolean adError — Error adUserDefined — User-defined data type adVariant — Variant adIDispatch — Pointer to IDispatch adIUnknown — Pointer to IUnknown adGUID — GUID adDate — Date adDBDate — DBDate

	adDBTime	DBTime
	adDBTimeStamp	DBTimestamp
	adBSTR	BSTR
	adChar	Char
	adVarChar	VarChar
	adLongVarChar	Long VarChar
	adWChar	Wide Char
	adVarWChar	VarWChar
	adLongVarWChar	Long VarWChar
	adBinary	Binary
	adVarBinary	VarBinary
	adLongVarBinary	Long VarBinary
	adChapter	Chapter
	adFileTime	FileTime
	adPropVariant	Variant Property
	adVarNumeric	VarNumeric
	adArray	Array
nDefinedSize	Size of the data for the field being added to the collection	
nFA	ADOFieldAttributeEnum An additional descriptor for the Field being added to the collection. Valid values are:	
	Value	Description
	adFldUnspecified	Unspecified attribute
	adFldMayDefer	MayDefer attribute
	adFldUpdatable	Updatable attribute
	adFldUnknownUpdatable	Unknown Updatable attribute
	adFldFixed	FldFixed attribute
	adFldIsNullable	IsNullable attribute
	adFldMayBeNull	FldMayBeNull attribute
	adFldLong	FldLong attribute

	adFldRowID	FldRowID attribute
	adFldRowVersion	RowVersion attribute
	adFldCacheDeferred	FldCacheDeferred attribute
	adFldIsChapter	FldIsChapter attribute
	adFldNegativeScale	FldNegativeScale attribute
	adFldIsRowURL	FldIsRowURL attribute
	adFldIsDefaultStream	FldIsDefaultStream attribute
	adFldIsCollection	FldIs Collection attribute
pvFieldValue	The Field's actual data in the form of a Pointer to a VARIANT.	

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] );
ADO_LoadVariant( pvValue, "8", "New ColumnData" );
ADO_FieldSet(1)->Append( "testfld1" ,adBSTR, 0 , adFldUnspecified, pvValue );
```

ADO_FieldSet(n)->Append15

Creates and appends a new field object to the ADO FieldSet.

An ADO Recordset object is composed of ADO FieldSet objects. Appending ADO Fields to ADO FieldSet objects comprise a mechanism for updating or retrieving information from a Data Provider. The Append15 function does NOT allow the user to add the data to this ADO Field object. It creates the ADO Field object in the ADO FieldSet collection, but does not add the data.

Syntax

```
ADO_FieldSet(n)->Append15( char* sNameString, ADODataTypeEnum nDT, long nDefinedSize,
ADOFieldAttributeEnum nFA );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sNameString	The name of the Field being added to the collection of fields.
nDT	ADODataTypeEnum The Data type of the field being added to the collection. Valid values are: Value — Description adEmpty — No data type specified

	adTinyInt	Tiny integer
	adSmallInt	Small integer
	adInteger	Integer
	adBigInt	Big integer
	adUnsignedTinyInt	Unsigned tiny integer
	adUnsignedSmallInt	Unsigned small integer
	adUnsignedInt	Unsigned integer
	adUnsignedBigInt	Unsigned integer
	adSingle	Single precision float
	adDouble	Double precision float
	adCurrency	Currency
	adDecimal	Decimal
	adNumeric	Numeric
	adBoolean	Boolean
	adError	Error
	adUserDefined	User-defined data type
	adVariant	Variant
	adIDispatch	Pointer to IDispatch
	adIUnknown	Pointer to IUnknown
	adGUID	GUID
	adDate	Date
	adDBDate	DBDate
	adDBTime	DBTime
	adDBTimeStamp	DBTimestamp
	adBSTR	BSTR
	adChar	Char
	adVarChar	VarChar
	adLongVarChar	Long VarChar
	adWChar	Wide Char

	adVarWChar	VarWChar
	adLongVarWChar	Long VarWChar
	adBinary	Binary
	adVarBinary	VarBinary
	adLongVarBinary	Long VarBinary
	adChapter	Chapter
	adFileTime	FileTime
	adPropVariant	Variant Property
	adVarNumeric	VarNumeric
	adArray	Array
nDefinedSize	Size of the data for the field being added to the collection	
nFA	ADOFieldAttributeEnum An additional descriptor for the Field being added to the collection. Valid values are:	
	Value	Description
	adFldUnspecified	Unspecified attribute
	adFldMayDefer	MayDefer attribute
	adFldUpdatable	Updatable attribute
	adFldUnknownUpdatable	Unknown Updatable attribute
	adFldFixed	FldFixed attribute
	adFldIsNullable	IsNullable attribute
	adFldMayBeNull	FldMayBeNull attribute
	adFldLong	FldLong attribute
	adFldRowID	FldRowID attribute
	adFldRowVersion	RowVersion attribute
	adFldCacheDeferred	FldCacheDeferred attribute
	adFldIsChapter	FldIsChapter attribute
	adFldNegativeScale	FldNegativeScale attribute
	adFldIsRowURL	FldIsRowURL attribute
	adFldIsDefaultStream	FldIsDefaultStream attribute

	adFldIsCollection	FldIs Collection attribute

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] );
ADO_FieldSet(1)->Append15( "testfld1", adBSTR, 0, adFldUn specified );
ADO_LoadVariant( pvValue, "8", "testfld1" );
ADO_FieldSet(1)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "New ColumnData" );
ADO_FieldSet(1)->PutValue( pvValue );
```

ADO_FieldSet(n)->CancelUpdate

Cancels changes made to the current or new row of an ADO Recordset object, or the ADO Fieldset collection of an ADO Record object, before calling the Update method.

Syntax

```
ADO_FieldSet(n)->CancelUpdate();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

None.

ADO_FieldSet(n)->Delete

Deletes an object from the Fields collection.

Takes the form of a Variant designating a Field object to delete. The Variant can be the name or ordinal position of the Field object.

Syntax

```
ADO_FieldSet(n)->Delete( VARIANT* pvObject );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pvObject	A pointer to a Variant. The variant depicts the field to remove.

Example

```
ADO_Recordset(1)->GetFields( ADOFieldSet[1] ); LoadVariant( pvValue, "2", "1" );
ADO_FieldSet(1)->Delete( pvValue );
ADOFieldSet.Release( 1 );
```

ADO_FieldSet(n)->GetCount

Number of ADO Field objects contained within the ADO FieldSet collection.

Syntax

```
ADO_FieldSet(n)->GetCount( long* pCount );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCount	A Pointer to a long containing the number of ADO Field objects in this ADO FieldSet Collection.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
ADO_FieldSet(0)->Refresh();
```

ADO_FieldSet(n)->GetItem

Retrieves an ADO Field object from this instance of the ADO FieldSet collection.

The result of the call is that a ADO Field object is brought back to be manipulated within the script. ADO Field retrieval is a part of the variablization process. In the example, sSSN is declared as a local variable and is a key element used in several calls to the data provider. This is variablized within the script and linked to a datapool earlier in the script.

Syntax

```
ADO_FieldSet(n)->GetItem( VARIANT* pvIndex, CAField* pField );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvIndex	This input parameter is used to pick the particular ADO Field instance from the ADO FieldSet collection.
pField	This output parameter is used to store away the ADO Field returned by this call.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "SSN" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", sSSN );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
ADORecordset.Release( 2, ADOBM );
```

ADO_FieldSet(n)->GetNewEnum

Creates the ADO IEnumField object.

In order to iterate through each ADO Field in an ADO FieldSet collection, an ADOIEnumField object is returned. The GetNewEnum call on the ADO FieldSet object creates the ADO IEnumField object allowing the enumeration to take place.

Syntax

```
ADO_FieldSet(n)->GetNewEnum( CAIEnumField* pADOIEnumField );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pADOIEnumField	ADO IEnumField object.

Example

```
ADO_FieldSet(0)->GetNewEnum( ADOIEnumField[0] );
while( ADO_IEnumField(0)->NextField( 1, 0, ADOField[0] ))
{
ADO_Field(0)->GetStatus( pLong );
ADOField.Release( 0 );
}
```

ADO_FieldSet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider.

Using the Refresh method on the ADO FieldSet collection has no visible effect.

Syntax

```
ADO_FieldSet(n)->Refresh();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
ADO_FieldSet(0)->Refresh();
```

ADO_FieldSet(n)->Resync

Synchronizes the values of a Record object's Fields collection with the data source.

The Count property is not affected by this method.

Syntax

```
ADO_FieldSet(n)->Resync( ADOResyncEnum nResync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nResync	ADOResyncEnum Resync option. Valid values are: Value — Description adResyncUnderlyingValues — Resync the underlying values only adResyncAllValues — Resync all values

Example

```
ADO_LoadVariant(pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADOField.Release( 0 );
ADO_FieldSet(0)->Update();
ADO_FieldSet(0)->Resync( adResyncAllValues );
```

ADO_FieldSet(n)->Update

Saves any changes you make to the ADO FieldSet collection of a Record object.

Syntax

```
ADO_FieldSet(n)->Update();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADOField.Release( 0 );
ADO_LoadVarian( pvValue, "8", "test_number" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "2", "15" );
ADO_Field(0)->PutValue( pvValue );
ADOField.Release( 0 );
ADO_FieldSet(0)->Update();
```

ADO_IEnumField(n)->NextField

Enumeration through collections of ADO Fields should be done very carefully, because in the example given below, the script checks the status of each of the different ADO Fields. In order to do more meaningful work, reset values of different ADO Fields, then break them out of the look and use the PutValue call to place new values into the ADO Field objects.

Syntax

```
ADO_IEnumField(n)->NextField( 1, 0, ADOField(0) );
```

Return Value

Parameters

Parameter	Description

n	An index to the object.
BFetched	Retrieve this property: 0 FALSE 1 TRUE
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE
ADOField[n]	The instance of the ADO Field object that we are interested in.

Example

```
ADO_FieldSet(0)->GetNewEnum( ADOIEnumField[0] );
while( ADO_IEnumField(0)->NextField( 1, 0, ADOField[0] ) )
{
ADO_Field(0)->GetStatus( pLong );
ADOField.Release( 0 );
}
ADOIEnumField.Release( 0 );
```

ADO_IEnum(n)->NextProperty

Enumeration through collections of properties should be done very carefully, because in the example below, we reset all of the properties to the same value. To reset different values, get rid of the loop and set each property individually.

Syntax

```
ADO_IEnum(n)->NextProperty( <bFetched>, <bRetrieved>, ADOProperty(0) );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
BFetched	Retrieve this property 0 FALSE 1 TRUE.
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE.
ADOProperty[#]	The ADO Property returned to be worked on.

Example

```
ADO_Connect(0)->GetProperties( ADOPropertySet[0] );
ADO_PropertySet(0)->GetNewEnum( ADOIEnum[0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOProperty[0] ) )
{
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
}
```

```
ADOIEnum.Release( 0 );
ADOPropertySet.Release( 0 );
```

ADO_IEnumParameter(n)->NextParameter

Enumeration through collections of ADO Parameters should be done very carefully, because in the example given below, the script checks different values of each of the different ADO Parameters. In order to do some more meaningful work, resetting values of different ADO Parameters then break them out of the look and use the PutValue call to place new values into the ADO Parameter objects.

Syntax

```
ADO_IEnumParameter(n)->NextParameter( 1, 0, ADOParameter[0] );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
BFetched	Retrieve this property 0 FALSE 1 TRUE
BRetrieved	Has this been retrieved: 0 FALSE 1 TRUE
ADOParameter[n]	The ADO Parameter being returned to be worked on

Example

```
ADO_Command(0)->GetParameters( ADOParameterSet[0] );
ADO_ParameterSet(0)->GetNewEnum( ADOIEnumParameter[0] );
while( ADO_IEnumParameter(0)->NextParameter( 1, 0,
ADOParameter[0] ) )
{
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADOParameter.Release( 0 );
}
ADOIEnumParameter.Release( 0 );
```

ADO_LoadVariant(n)

Loads the value sValue, of type sType, into the Variant structure.

Syntax

```
ADO_LoadVariant ( <VariantName>, "<Type>", "<Value>");
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
VariantName	The variable that is being set up by this method.
Type	The VT_TYPE of the data. The VT_TYPE is the key to the information contained within any Variant data structure. It reveals which of the data elements within the variant structure contains data.
sValue	The actual data that is placed into whatever VT_TYPE is called for.
ADOCommand[#]	A pointer to a specific instance of ADO Command.

Example

```
ADO_Connect(0)->GetProperties( ADOPropertySet[0] );
ADO_PropertySet(0)->GetNewEnum( ADOIEnum[0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOProperty[0] ) )
{
ADO_Property(0)
```

ADO_Parameter(n)->AppendChunk

A special data handling method that writes data in chunks to the Parameter object.

This is especially useful when memory is limited, since you can use this method to manipulate long values in manageable chunks. It may take numerous calls to AppendChunk to completely write the data to the appropriate object.

When writing data values using ADO, the datatype being used as the parameter with the data value is often a VARIANT datatype. The first call to AppendChunk writes data to the parameter and overwrites any existing data. Subsequent calls add to the data. Note that if you append data to one parameter, then manipulate another parameter in the same record, ADO assumes you are finished with the first parameter. If you then attempt to append data to the first parameter, the existing data will be overwritten.

The AppendChunk call will be preceded immediately in the script by a call to ADO_LoadVariant.

You can use the AppendChunk method in the Attributes property of a parameter object if the adFldLong bit in the Attributes property is set to true.

Syntax

```
ADO_Parameter(n)->AppendChunk( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description

n	An index to the object.
pvValue	The Variant containing the chunk of data to send to the ADO_Parameter object instance.

Example

```
ADO_Parameter(0)->PutType( adBSTR );
ADO_LoadVariant( pvValue, "8", "a big chunk of data" );
BeginCheckpoint("ADODataBinder::AppendChunk");
ADO_Parameter(0)->AppendChunk( pvValue );
EndCheckpoint("ADODataBinder::AppendChunk");
ADO_LoadVariant( pvValue, "8", "some more data" );
BeginCheckpoint("ADODataBinder::AppendChunk");
ADO_Parameter(0)->AppendChunk( pvValue );
EndCheckpoint("ADODataBinder::AppendChunk");
```

ADO_Parameter(n)->GetAttributes

Retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->GetAttributes( long* pAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pAttributes	Pointer to a long containing the attribute or Direction value of the Parameter object.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
```

ADO_Parameter(n)->GetDirection

Retrieves the value contained within the Direction property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->GetDirection( long* pDirection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pDirection	Pointer to a long containing the attribute or Direction value of the Parameter object.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
```

ADO_Parameter(n)->GetName

Retrieves the value contained within the Name property of this instance of the ADO Parameter object.

GetName is read/write for Parameter objects that haven't been appended to the Parameters collection. It is read-only for appended Parameter objects and all other objects. Note that within a collection, names do not have to be unique.

Syntax

```
ADO_Parameter(n)->GetName( CLoadString& sName );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sName	A CLoadString value being returned with the name of the Parameter.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
ADO_Parameter(0)->GetPrecision( pUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
ADO_Parameter(0)->GetValue( pvValue );
```

ADO_Parameter(n)->GetNumericScale

Retrieves the value contained within the NumericScale property of this instance of the ADO Parameter object.

Returns a Byte value indicating the number of decimal places to which numeric values is resolved. The NumericScale property is read/write.

Syntax

```
ADO_Parameter(n)->GetNumericScale( unsigned char* pucNumericScale );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pucNumericScale	The address of an unsigned character.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(0)->GetPrecision( &cUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
```

ADO_Parameter(n)->GetPrecision

Retrieves the value contained within the Precision property of this instance of the ADO Parameter object.

Returns a Byte value showing the maximum number of digits used to represent values for a numeric Parameter object. The Precision property is read/write.

Syntax

```
ADO_Parameter(n)->GetPrecision( unsigned char* pucPrecision );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pucPrecision	The address of an unsigned character.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(0)->GetPrecision( &cUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
```

ADO_Parameter(n)->GetSize

Retrieves the value contained within the Size property of this instance of the ADO Field object.

GetSize indicates the maximum size of a Parameter object in bytes or characters. You can use it to determine the maximum size for values of Parameter object's Value property.

If the data type specified for a Parameter object is of variable length, set the object's Size property before appending it to the Parameters collection. If you do not, an error occurs.

Syntax

```
ADO_Parameter(n)->GetSize( long* pSize );
```

Return Value

Parameter

Parameter	Description
n	An index to the object.
pSize	A pointer to a long containing the maximum size of the parameters data.

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( pUChar );
ADO_Parameter(0)->GetPrecision( pUChar );
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
ADO_Parameter(0)->GetValue( pvValue );
```

ADO_Parameter(n)->GetValue

Returns data from ADO Parameter objects and from parameter values with ADO Parameter objects.

Syntax

```
ADO_Parameter(n)->GetValue( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	Pointer to a variant used to retrieve data from the Parameter object.

Example

```
ADO_Parameter(2)->PutSize( 8 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
```

ADO_Parameter(n)->PutAttributes

This method call retrieves the value contained within the Attributes property of this instance of the ADO Parameter object.

PutAttributes is read/write. It's value can be the sum of one or more ParameterAttributesEnum values. The default is adParamSigned. The following are ParameterAttributesEnum values:

- adParamSigned
- adParamNullable
- adParamLong

Syntax

```
ADO_Parameter(n)->PutAttributes( ParameterAttributesEnum nAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAttributes	Special enumerated data type used.

Example

```
ADO_Command(0)->Execute( pvValue, pvSource, -1,
ADORecordset[0] );
ADO_Parameter(0)->PutAttributes( adParamLong );
ADO_Parameter(0)->PutType( adBSTR );
```

ADO_Parameter(n)->PutDirection

Indicates Parameter type: input, output, input and output, or the return value from a stored procedure.

This method call sets the value contained within the Direction property of this instance of the ADO Parameter object.

Syntax

```
ADO_Parameter(n)->PutDirection( ADOParameterDirectionEnum nDirection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nDirection	ADOParameterDirectionEnum Value — Description adParamUnknown — Unknown parameter status adParamInput — Parameter is input value adParamOutput — Parameter is output value adParamInputOutput — Parameter is input/output value adParamReturnValue — Parameter is return value

Example

```
ADO_Parameter(0)->GetAttributes( pLong );
ADO_Parameter(0)->GetName( sLoadStr );
ADO_Parameter(0)->GetDirection( pLong );
ADO_Parameter(0)->GetNumericScale( &cUChar );
ADO_Parameter(1)->PutDirection( adParamOutput );
```

ADO_Parameter(n)->PutName

Sets the value contained within the Name property of this instance of the ADO Parameter object.

PutName is read/write for Parameter objects that are not appended to the Parameters collection. It is read-only for appended Parameter objects and all other objects. Note that names do not have to be unique within a collection.

Syntax

```
ADO_Parameter(n)->PutName( char* sNameString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sNameString	A string containing the name of the parameter.

Example

```
ADO_Parameter(2)->PutAttributes( 128 );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
```

ADO_Parameter(n)->PutNumericScale

Sets the value contained within the NumericScale property of this instance of the ADO Parameter object.

Sends a byte value indicating the number of decimal places to which numeric values are resolved. The NumericScale property is read/write.

Syntax

```
ADO_Parameter(n)->PutNumericScale( unsigned char nucNumericScale );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nucNumericScale	This byte value sets the numeric scale. Format: 0x##

Example

```
ADO_Parameter(2)->PutAttributes( adParamLong );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
```

ADO_Parameter(n)->PutPrecision

Sets the value contained within the Precision property of this instance of the ADO Parameter object.

Sends a byte value showing the maximum number of digits used to represent values for a numeric ADO Parameter object. The Precision property is read/ write.

Syntax

```
ADO_Parameter(n)->PutPrecision( unsigned char nucPrecision );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nucPrecision	This byte value sets the precision. Format: 0x##

Example

```
ADO_Parameter(2)->PutAttributes( adParamLong );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
```

ADO_Parameter(n)->PutSize

Retrieves the value contained within the Size property of this instance of the ADO Parameter object.

Specifies the maximum size of a Parameter object in bytes or characters. You can use it to determine the maximum size for values of a Parameter object's Value property.

If the data type specified for a Parameter object is of variable length, set the object's Size property before appending it to the Parameters collection. If you do not, an error occurs.

Syntax

```
ADO_Parameter(n)->PutSize( long nSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nSize	A long integer representation of the size of the parameter.

Example

```
ADO_Parameter(0)->GetSize( pLong );
ADO_Parameter(0)->GetType( pLong );
ADO_Parameter(1)->PutSize( 4 );
ADO_Parameter(1)->PutType(adInteger );
```

ADO_Parameter(n)->PutType

Sets the value contained within the Type property of this instance of the ADO Parameter object.

PutType is read/write when each of the following conditions are present:

- it is on a new Parameter object
- the new Parameter object has been appended to a Record's Fields collection
- the Parameter's Value property has been specified
- the data provider has added the new Parameter (using the Parameters collection's Update method)

Syntax

```
ADO_Parameter(n)->PutType( ADODataTypeEnum nType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nType	ADODataTypeEnum The datatype of the field element. Valid values are: Value — Description adEmpty — No data type specified adTinyInt — Tiny integer adSmallInt — Small integer adInteger — Integer adBigInt — Big integer adUnsignedTinyInt — Unsigned tiny integer adUnsignedSmallInt — Unsigned small integer adUnsignedInt — Unsigned integer adUnsignedBigInt — Unsigned integer adSingle — Single precision float adDouble — Double precision float adCurrency — Currency adDecimal — Decimal adNumeric — Numeric adBoolean — Boolean

	adError	Error
	adUserDefined	User-defined data type
	adVariant	Variant
	adIDispatch	Pointer to IDispatch
	adIUnknown	Pointer to IUnknown
	adGUID	GUID
	adDate	Date
	adDBDate	DBDate
	adDBTime	DBTime
	adDBTimeStamp	DBTimestamp
	adBSTR	BSTR
	adChar	Char
	adVarChar	VarChar
	adLongVarChar	Long VarChar
	adWChar	Wide Char
	adVarWChar	VarWChar
	adLongVarWChar	Long VarWChar
	adBinary	Binary
	adVarBinary	VarBinary
	adLongVarBinary	Long VarBinary
	adChapter	Chapter
	adFileTime	FileTime
	adPropVariant	Variant Property
	adVarNumeric	VarNumeric
	adArray	Array

Example

```
ADO_Parameter(2)->PutAttributes( 128 );
ADO_Parameter(2)->PutDirection( adParamOutput );
ADO_Parameter(2)->PutName( "testParam" );
ADO_Parameter(2)->PutNumericScale( 0x03 );
ADO_Parameter(2)->PutPrecision( 0x02 );
ADO_Parameter(2)->PutSize( 4 );
ADO_Parameter(2)->PutType( adDouble );
```

```
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
```

ADO_Parameter(n)->PutValue

Sets the value contained within the Value property of this instance of the ADO Parameter object.

PutValue can be used to set or return data from ADO Parameter objects, parameter values with ADO Parameter objects, or property settings with Property objects.

Syntax

```
ADO_Parameter(n)->PutValue( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	The value being loaded into this parameter.

Example

```
ADO_Parameter(2)->PutSize( 8 );
ADO_Parameter(2)->PutType( adDouble );
ADO_LoadVariant( pvValue, "5", "5.6 " );
ADO_Parameter(2)->PutValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
ADO_Parameter(3)->GetValue( pvValue );
```

ADO_ParameterSet(n)->Append

Appends a ADO Parameter object to the collection of ADO Parameters.

ADO Parameters are created and given a value using CreateParameter calls, then the ADO Parameters are Appended to the ADO ParameterSet container.

Syntax

```
ADO_ParameterSet(n)->Append( CAParameter* pParam );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pParam	ADO Parameter object being added to this collection.

Example

```
ADO_LoadVariant( pvValue, "8", "ParameterData0" );
ADO_Command(0)->CreateParameter( "Param3", adInteger, adParamInput,
                                 0, pvVAlue, ADOParameter[0] );
ADO_LoadVariant( pvValue, "8", "ParameterData1" );
ADO_Command(0)->CreateParameter( "Param4", adInteger, adParamInput,
                                 0, pvValue, ADOParameter[1] );
ADO_ParameterSet(0)->Append( ADOParameter[0] );
ADO_ParameterSet(0)->Append( ADOParameter[1] );
```

ADO_ParameterSet(n)->Delete

Deletes an ADO Parameter object from the ADO ParameterSet collection.

Takes the form of a Variant designating an ADO Parameter object to delete. The Variant can be the name or ordinal position of the ADO Parameter object.

Syntax

```
ADO_ParameterSet(n)->Delete( VARIANT* pvIndex );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvIndex	A pointer to a variant containing information describing the ADO Parameter to be deleted from the ADO ParameterSet collection.

Example

```
ADO_LoadVariant( pvValue, "2", "3" );
BeginCheckpoint("ADOParameterSet::Delete");
ADO_ParameterSet(0)->Delete( pvValue );
EndCheckpoint("ADOParameterSet::Delete");
```

ADO_ParameterSet(n)->GetCount

The method returns the number of ADO Parameter objects contained within the ADO ParameterSet collection.

Syntax

```
ADO_ParameterSet(n)->GetCount( long* pnCount );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pnCount	A pointer to a long containing the number of ADO Parameter objects in this ADO ParameterSet Collection.

Example

```
ADO_ParameterSet(0)->GetCount( pLong );
ADO_ParameterSet(0)->Refresh();
```

ADO_ParameterSet(n)->GetItem

Locates a specific ADO Parameter in the ADO ParameterSet collection.

An ADO ParameterSet collection is an array of ADO Parameter objects. GetItem indexes through the array to locate a specific object.

Syntax

```
ADO_ParameterSet(n)->GetItem( VARIANT* pvIndex, CAParameter* pParameter );
```

Return Value

Parameters

Parameter	Description	
n	An index to the object.	
pvIndex	The variant contains information about the parameter to retrieve from the collection.	p QAL_Attribute=Parameter_Description class=code>The retrieved parameter.

Example

```
BeginCheckpoint("ADOParameterSet::GetItem");
ADO_ParameterSet(0)->GetItem( pvValue, ADOParameter[0] );
EndCheckpoint("ADOParameterSet::GetItem");
```

ADO_ParameterSet(n)->GetNewEnum

Creates the ADO IEnumParameter object.

In order to iterate through all of the ADO Parameters in an ADO ParameterSet collection, an ADOIEnumParameter object is returned. The GetNewEnum call on the ADO ParameterSet object creates the ADO IEnumParameter object allowing the enumeration to take place.

Syntax

```
ADO_ParameterSet(n)->GetNewEnum( CAIEnumParameter* pADOIEnumParameter );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pADOIEnumParameter	ADO IEnumParameter object.

Example

```
ADO_ParameterSet(0)->GetNewEnum( ADOIEnumParameter[0] );
while( ADO_IEnumParameter(0)->NextParameter( 1, 0, ADOParameter[0] ));
{
ADO_Parameter(0)->GetStatus( pLong );
ADOParameter.Release( 0 );
}
```

ADO_ParameterSet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider.

Using the Refresh method on the ADO ParameterSet collection has no visible effect.

Syntax

```
ADO_ParameterSet(n)->Refresh();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_ParameterSet(0)->GetCount( pLong );
ADO_ParameterSet(0)->Refresh();
```

ADO_Property(n)->GetAttributes

Describes column characteristics by setting or returning a Long value.

The value indicates characteristics of the table represented by the Column object. It can be a combination of ColumnAttributesEnum constants. The default value is zero (0).

Syntax

```
ADO_Property(n)->GetAttributes( long* pAttributes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pAttributes	A pointer to a long integer containing the value of the Attributes property.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );

/* Type:8 - VT_BSTR Data: Master */

ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetName

Retrieves the value of the Name attribute of this instance of the Property object.

Syntax

```
ADO_Property(n)->GetName( CLoadString& sName );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sName	Value of the Name property for this instance of the ADO Property.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
```

```
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetType

Indicates a property's type as conveyed as a DataTypeEnum.

Syntax

```
ADO_Property(n)->GetType( long* pType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pType	A pointer to a long containing the DataTypeEnum value for the property.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->GetValue

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

Note: You can use the Value property to set and return long binary data.

Syntax

```
ADO_Property(n)->GetValue( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pvValue	A Pointer to a variant in which the value of this property will be returned.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_Property(n)->PutAttributes

Sets the value contained within the Attributes property of this instance of the ADO Property object.

Syntax

```
ADO_Property(n)->PutAttributes( ADOPropertyAttributesEnum nAts );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAts	ADOPropertyAttributesEnum The attribute type as PropertyAttributesEnum. Valid values are: Value — Description adPropNotSupported — Indicates that the property is not supported by the provider adPropRequired — Indicates that the user must specify a value for this property before the data source is initialized adPropOptional — Indicates that the user does not need to specify a value for this property before the data source is initialized adPropRead — Indicates that the user can read the property adPropWrite — Indicates that the user can

	set the property

n An index to the object.
nAts A long integer value that should reflect one of the following:
adPropNotSupported (value=0), adPropRequired (value=1), adPropOptional (value=2), adPropRead (value=512), adPropWrite (value=1024), or a combination of them.

ADO_Property(n)->PutValue

Sets or returns data from Field objects, parameter values with Parameter objects, or property settings with Property objects.

Note: You can use the Value property to set and return long binary data.

Syntax

```
ADO_Property(n)->PutValue( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	The pointer to the VARIANT retrieves the Value of the property that is in the get, and the value is set in the Put call.

Example

```
ADO_Property(0)->GetAttributes( pLong );
ADO_Property(0)->GetName( sLoadStr );
ADO_Property(0)->GetType( pLong );
ADO_Property(0)->GetValue( pvValue );
ADO_LoadVariant( pvValue, "8", "A Test String" );
ADO_Property(0)->PutValue( pvValue );
ADO_LoadVariant( pvValue, "8", "Master" );
ADO_Property(0)->PutValue( pvValue );
ADOProperty.Release( 0 );
```

ADO_PropertySet(n)->GetCount

Returns the number of ADO Property objects contained within the ADO PropertySet collection.

Syntax

```
ADO_PropertySet(n)->GetCount( long* pnCount );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pnCount	A pointer to a long containing the number of ADO Property objects in this ADO PropertySet Collection.

Example

```
ADO_PropertySet(0)->GetCount( pLong );
ADO_PropertySet(0)->Refresh();
```

ADO_PropertySet(n)->GetItem

Retrieves a specific ADO Property in the ADO PropertySet collection.

An ADO PropertySet collection is an array of ADO Property objects. GetItem indexes through the array to locate a specific object.

Syntax

```
ADO_PropertySet(n)->GetItem( VARIANT* pvIndex, CAProperty* pProperty );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvIndex	A pointer to a variant describing the property to be retrieved.
pProperty	An instance of an ADO Property object.

Example

```
ADO_LoadVariant( pvValue, "2", "3" );
ADO_PropertySet(0)->GetItem( pvValue, ADOProperty[0] );
```

ADO_PropertySet(n)->GetNewEnum

Creates the ADO IEnum object.

In order to iterate through all of the ADO Propertys in an ADO PropertySet collection, an ADOIEnum object is returned. The GetNewEnum call on the ADO PropertySet object creates the ADO IEnum object allowing the enumeration to take place.

Syntax

```
ADO_PropertySet(n)->GetNewEnum( CAIEnumProperty* pADOIEnum );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pADOIEnum	ADO IEnum object.

Example

```
ADO_PropertySet(0)->GetNewEnum( ADOIEnum [0] );
while( ADO_IEnum(0)->NextProperty( 1, 0, ADOProperty[0] ))
{
ADO_Property(0)->GetStatus( pLong );
ADOProperty.Release( 0 );
}
```

ADO_PropertySet(n)->Refresh

Updates the objects in a collection to reflect objects available from, and specific to, the provider.

Using the Refresh method on the ADO PropertySet collection has no visible effect.

Syntax

```
ADO_PropertySet(n)->Refresh();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_PropertySet(0)->GetCount( pLong );
ADO_PropertySet(0)->Refresh();
```

ADO_Record(n)->Cancel

Cancels execution of a pending, asynchronous method call.

Syntax

```
ADO_Record(n)->Cancel();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Record(1)->CopyRecord( "Home", "Away", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
```

ADO_Record(n)->Close

Use to close a Recordset, Record, or Stream object.

Any associated data or exclusive access you may have had to the data through this particular object is released. You can reopen the object later using the Open method.

Syntax

```
ADO_Record(n)->Close();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Record(1)->CopyRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->CopyRecord

Copies a file or directory (including its contents) to another location.

Tip: Ensure that the values of Source and Destination are not identical or you will receive a run-time error. One of the server, path, or resource names must differ.

All subdirectories are copied recursively unless adCopyNonRecursive is specified. In a recursive operation, Destination must not be a subdirectory of Source; otherwise, the operation is not able to finish.

Insert your product name (QALoad , for example)'s implementation of the CopyRecord method makes the call through to the CopyRecord method within the ADO Record object.

Note that the CopyRecordOptionsEnum is often in the form of a number. This occurs when the CopyRecordOptionsEnum is formed from a combination of values.

Syntax

```
ADO_Record(n)->CopyRecord( char* sSourceString, char* sDestString, char* sUserString, char*
sPasswordString, ADOCopyRecordOptionsEnum nOptions, short bAsync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSourceString	String value containing a URL that specifies the entity that is to be copied.
sDestString	String value containing a URL that specifies the location to which the Source is copied.
sUserString	This is the user name used to determine whether a particular user has permission to use this information.
sPasswordString	A String value. The password for the particular user to verify that the user has permission to perform the operation.
nOptions	ADOCopyRecordOptionsEnum Copy options. Valid values are: Value — Description adMoveUnspecified — Unspecified copy record option adMoveOverWrite — Overwrite record at location adMoveDontUpdateLinks — Don't updatelinks when record copied adMoveAllowEmulation — Allow emulation
bAsync	If this is an asynchronous operation.

Example

```
ADO_Record(1)->CopyRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->DeleteRecord

Deletes a file or directory and all its subdirectories.

After this method is finished, any operations on the file or directory represented by this Record could fail. Close the Record after calling this method.

Insert your product name, QALoad, for example. The DeleteRecord method makes the call through to the DeleteRecord method within the ADO Record object.

Syntax

```
ADO_Record(n)->DeleteRecord( char* sSourceString, short bAsync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSourceString	A string value that contains a URL identifying the entity to be deleted, for example, the file or directory.
bAsync	Is this an asynchronous call (-1 TRUE, 0 FALSE)

Example

```
ADO_Record(1)->DeleteRecord( "\\\\QAServer\\Temp\\GeoffR", 0 );
ADO_Record(1)->Cancel();
ADO_Record(1)->CLose();
```

ADO_Record(n)->GetActiveConnection

Determines the ADO Connect object over which the specified ADO Record object executes.

Syntax

```
ADO_Record(n)->GetActiveConnection( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	A Pointer to a Variant containing the ADO Connect object.

Example

```
LoadVariant( pvValue, ADOConnect[1] );
ADO_Record(1)->GetActiveConnection( pvValue );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->GetChildren

Returns an ADO Recordset in the form of a Pointer to an ADO Recordset object.

The rows represent the files and subdirectories in the directory represented by this Record.

Syntax

```
ADO_Record(n)->GetChildren( CARecordSet* pRecordSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pRecordSet	This is the information retrieved from this call. The GetChildren call retrieves the data into a ADO Recordset pointer.

Example

```
ADO_Record(1)->GetChildren( ADORecordset[1] );
ADO_Record(1)->Close();
```

ADO_Record(n)->GetFields

Contains all the Field objects of an ADO Recordset or ADO Record object.

Insert your product name, QALoad , for example. The GetFields method takes care of making the call through to the GetFields method within the ADO Record object. In this call, the ADO Fields object that is returned is wrapped within the ADO FieldSet object.

Syntax

```
ADO_Record(n)->GetFields( CAFieldSet* pFieldSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pFieldSet	Set of fields that compose the record.

Example

```
ADO_Record(1)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "dsn_name" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetValue( pvValue ); /* Type: 8 - VT_BSTR Data: FOCFG */
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_Record(1)->MoveNext();
ADO_Record(1)->GetEOF( pVTBOOL );
ADO_Record(1)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "dsn_name" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_Field(0)->GetValue( pvValue ); /* Type: 8 - VT_BSTR Data: FOCRP */
```

```
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
```

ADO_Record(n)->GetMode

Sets or returns the access permissions being used on the current connection by the provider.

Note that you can only set this property when the Connection object is closed.

Syntax

```
ADO_Record(n)->GetMode( long* pMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pMode	Retrieves the ConnectionModeEnum from the call and converts that to a long* to be returned to the script.

Example

```
ADO_Record(1)->GetMode( pLong );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->GetParentURL

Sets the current value of the source property for this instance of the actual ADO Command object.

This property depends on which source is used to open the Record object.

Syntax

```
ADO_Record(n)->GetParentURL( CLoadString& sParent );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sParent	CLoadString holding the parent URL retrieved by the call.

Example

```
ADO_Record(1)->PutSource( "\\\\QAServer\\MyDirectory" );
ADO_Record(1)->GetParentURL( sLoadStr );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->GetRecordType

Checks the contents of the ADO RecordType property for this instance of ADO Record object.

It returns the RecordTypeEnum in a pointer to a long.

Syntax

```
ADO_Record(n)->GetRecordType( RecordTypeEnum *pRecordType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pLong	Pointer to a long containing a RecordTypeEnum value.
pRecordType	Contains the following values ! adSimpleRecord (value=0) ! adCollectionRecord (value=1) ! adStructDoc (value=2)

Example

```
ADO_Record(1)->PutSource( "\\\\QAServer\\MyDirectory" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->GetSource

Indicates the entity represented by the ADO Record object.

The GetSource method retrieves the current value of the source property of the ADO Record object.

Syntax

```
ADO_Record(n)->GetSource( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	Variant holding the value of the source of the Record object.

Example

```
ADO_Record(1)->GetSource( pvValue );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 );
```

ADO_Record(n)->GetState

Determines the state of a given ADO Record object at any time.

Syntax

```
ADO_Record(n)->GetState( long* pState );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pState	Pointer to a long integer holding the state of the Record object.

Example

```
ADO_Record(1)->GetState( pLong );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
ADORecord.Release( 1 )
```

ADO_Record(n)->MoveRecord

Moves a file or a directory and its contents to another location.

A run-time error occurs if the values of Source and Destination are the same. At least one of the server, path, and resource names must differ.

All hypertext links are updated unless otherwise specified by Options. If an existing file or directory is identified, this method fails unless you specify adMoveOverWrite.

Note that the MoveRecordOptionsEnum is often in the form of a number. This occurs when the MoveRecordOptionsEnum is formed from a combination of values.

Syntax

```
ADO_Record(n)->MoveRecord( char* sSourceString, char* sDestString, char* sUserString, char* sPasswordString, ADOMoveRecordOptionsEnum nOptions, short bAsync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSourceString	String value containing a URL that specifies the entity that is to be copied.
sDestString	String value containing a URL that specifies the location to which the Source is copied.
sUserString	This is the user name used to determine whether a particular user has permission to use this information.
sPasswordString	A String value. The password for the particular user to verify that the user has permission to perform the operation.
nOptions	*ADOMoveRecordOptionsEnum* Move options. Valid values are: Value — Description adMoveUnspecified — Unspecified move record option adMoveOverWrite — Overwrite record at location adMoveDontUpdateLinks — Don't update links when record moved adMoveAllowEmulation — Allow emulation
bAsync	If this is an asynchronous operation.

Example

```
ADO_Record(1)->MoveRecord( "C:\\Home", "D:\\Away", "sa", "sa", adCopyOverWrite, -1 );
ADO_Record(1)->Cancel();
ADO_Record(1)->Close();
```

ADO_Record(n)->Open

Makes the call through to the Open method within the ADO Record object to open an existing ADO Record object, or create a new file or directory.

If the entity represented by the Record object can't be accessed with a URL, the values of the ParentURL property and the field accessed with the adRecordURL constant are null.

Note: The two SetupVariantValue calls must be present. They also present opportunities for variablization of the scripts.

Syntax

```
ADO_Record(n)->Open( VARIANT* pvSource, VARIANT* pvActiveConnection, ADOConnectModeEnum
nMode, ADORecordCreateOptionsEnum nCreateOptions, ADORecordOpenOptionsEnum nOpenOptions,
char* sUserName, char* sPassword );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a variant that may represent the URL of the entity to be represented by this ADO Record object, an ADO Command, an open ADO Recordset or another ADO Record object, a string containing a SQL SELECT statement, or a table name.
pvActiveConnection	A pointer to a variant that represents the connect string or open ADO Connect object.
nMode	ADOConnectModeEnum A ConnectModeEnum value, whose default value is adModeUnknown, that specifies the access mode for the resultant Record object. Valid values are: Value — Description adModeUnknown — Unknown connection mode adModeRead — Read-only mode adModeWrite — Write-only mode adModeReadWrite — Read-write mode adModeShareDenyRead — Exclusive read mode adModeShareDenyWrite — Exclusive write mode adModeShareExclusive — Exclusive read-write mode adModeShareDenyNone — Non-exclusive mode adModeRecursive — Recursive mode
nCreateOptions	ADORecordCreateOptionsEnum A RecordCreateOptionsEnum value, whose default value is adFailIfNotExists, that specifies whether an existing file or directory should be opened, or a new file or directory should be created. Valid values are: Value — Description adFailIfNotExists — Default. Results in a run-time error if Source points to a non-existent node

	adCreateNonCollection	Creates a new Record of type adSimpleRecord
	adCreateCollection	Creates a new Record at the node specified by Source parameter, instead of opening an existing Record
	adOpenIfExists	Modifies the creation flags adCreateCollection, adCreateNonCollection, and adCreateStructDoc
	adCreateOverwrite	Modifies the creation flags adCreateCollection, adCreateNonCollection, and adCreateStructDoc
	adCreateStructDoc	Creates a new Record of type adStructDoc, instead of opening an existing Record

nOpenOptions

ADORecordOpenOptionsEnum

A RecordOpenOptionsEnum value, whose default value is adOpenRecordUnspecified, that specifies options for opening the ADO Record. Valid values are:

Value	Description
adOpenRecordUnspecified	Default. Indicates no options are specified
adOpenAsync	Indicates that the Record object is opened in asynchronous mode
adDelayFetchStream	Indicates to the provider that the default stream associated with the Record need not be retrieved initially
adDelayFetchFields	Indicates to the provider that the fields associated with the Record need not be retrieved initially, but can be retrieved at the first attempt to access the field
adOpenExecuteCommand	Indicates that the Source string contains command text that should be executed
adOpenOutput	Indicates that if the source points to a node that contains an executable script, then the opened Record will contain the results of the executed

	script
sUserName	A String value that contains the user ID that, if needed, authorizes access to Source.
sPassword	A String value that contains the password that, if needed, verifies UserName.

Example

```
ADO_LoadVariant( pvSource, "8", "\\\\QAServer\\ MyDirectory" );
ADO_LoadVariant( pvValue, "8", "\\\\QAServer\\ BossDirectory" );
ADO_Record(1)->Open( pvSource, pvValue, adModeReadWrite, adCreateCollection, adOpenOutput,
"sa", "sa" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutActiveConnection

May contain a connection string or reference to an open ADO Connect object. PutActiveConnection is read/write when the ADO Record object is closed.

When the ADO Record object is open and contains a reference to an open ADO Connect object, PutActiveConnection is read-only.

Syntax

```
ADO_Record(n)->PutActiveConnection( char* sConnectionString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sConnectionString	A Connection string.

Example

```
ADO_Record(1)->PutActiveConnection( "DSN=QAServer; UID=sa; PWD=sa" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutMode

Sets the access permissions being used on the current connection by the provider.

You can only set this property when the ADO Connect object is closed.

Syntax

```
ADO_Record(n)->PutMode( ADOConnectModeEnum nMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nMode	ADOConnectModeEnum A ConnectModeEnum value, whose default value is adModeUnknown, that specifies the access mode for the resultant Record object. Valid values are: Value — Description adModeUnknown — Unknown connection mode adModeRead — Read-only mode adModeWrite — Write-only mode adModeReadWrite — Read-write mode adModeShareDenyRead — Exclusive read mode adModeShareDenyWrite — Exclusive write mode adModeShareExclusive — Exclusive read-write mode adModeShareDenyNone — Non-exclusive mode adModeRecursive — Recursive mode

Example

```
ADO_Record(1)->PutMode( adModeShareDenyNone );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutRefActiveConnection

Specifies the ADO Connect object to be affected by the specified ADO Record object.

The Argument being passed to this call is an ADO Connect. This is resolved through the ADOConnect[#] operator call. In the example below, the ADO Record is associating itself with ADO Connect object index 2.

Syntax

```
ADO_Record(n)->PutRefActiveConnection( CAConnect* pConnect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pConnect	An existing instance of an ADO Connect object.

Example

```
ADO_Record(1)->PutRefActiveConnection( ADOConnect[2] );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutRefSource

Sets the current value of the source property for this instance of the actual ADO Command object.

The Source property must refer to an object existing within the scope of that ADO Connect.

The Source property returns the Source argument of the ADO Record object Open method. It can contain an absolute or relative URL string. An absolute URL can be used without setting the ActiveConnection property to directly open the ADO Record object. An implicit ADO Connect object is created in this case.

Syntax

```
ADO_Record(n)->PutRefSource( VARIANT* pvSource );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvSource	A VARIANT representation of a source.

Example

```
ADO_Record(1)->PutRefSource( pvSource );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Record(n)->PutSource

Sets the current value of the source property for this instance of the actual ADO Command object.

The Source property must refer to an object existing within the scope of that ADO Connect.

The Source property returns the Source argument of the ADO Record object Open method. It can contain an absolute or relative URL string. An absolute URL can be used without setting the ActiveConnection property to directly open the ADO Record object. An implicit ADO Connect object is created in this case.

Syntax

```
ADO_Record(n)->PutSource( char* sSourceString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSourceString	A string representation of an absolute or relative URL.

Example

```
ADO_Record(1)->PutSource( "\\\\QAServer\\Development Home.htm" );
ADO_Record(1)->GetRecordType( pLong );
ADO_Record(1)->Close();
```

ADO_Recordset(n)->AddNew

O

Creates a new record for an updatable ADO Recordset object.

After AddNew is called, the new record becomes current and remains so after you call the Update method. If the ADO Recordset object doesn't support bookmarks, you may not be able to access the new record after moving to another record. You may need to call the Requery method to make the new record accessible.

In the example below, an empty row is being added to the end of the just opened ADO Recordset.

Syntax

```
ADO_Recordset(n)->AddNew( VARIANT* pvFieldList, VARIANT* pvValues );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvFieldList	A pointer to a Variant containing an Array of fields that compose the ADO Recordset.
pvValues	A pointer to an array of values corresponding to the array of fields.

Example

```
ADO_LoadVariant( pvSource, "8", "select sPhone, sExtension,
                 sDescription" ", iRecordID, sStudentID from PHONE
                 where""sStudentID='S123456'" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic,
                         adLockBatchOptimistic, -1 );
ADO_Recordset(27)->GetEOF( pVTBOOL );
```

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Recordset(27)->AddNew( pvSource, pvValue );
```

ADO_Recordset(n)->Cancel

Cancels execution of a pending, asynchronous method call.

Syntax

```
ADO_Recordset(n)->Cancel();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_LoadVariant( pvSource, "8", "SELECT * FROM test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Cancel();
```

ADO_Recordset(n)->CancelBatch

Cancels any pending updates in an ADO Recordset that is in batch update mode.

If the ADO Recordset is in immediate update mode and you call CancelBatch without adAffectCurrent, an error results.

Syntax

```
ADO_Recordset(n)->CancelBatch( ADOAffectEnum nAffect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAffect	ADOAffectEnum The recordset affect enumerator. Valid values include:

	Value	Description
	adAffectCurrent	Affects current only
	adAffectGroup	Affects group
	adAffectAll	Affects all
	adAffectAllChapters	Affects all chapters

Example

```
ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic, adLockBatchOptimistic, -1 );
ADO_Recordset(27)->CancelBatch(adAffectCurrent );
```

ADO_Recordset(n)->CancelUpdate

Cancels any changes made to the current row or discards a new row of an ADO Recordset object before calling the Update method.

You can only cancel changes to a current or new row after calling the Update method under the following conditions:

- The changes are part of a transaction that you can roll back with the RollbackTrans method.
- The changes are part of a batch update.

Syntax

```
ADO_Recordset(n)->CancelUpdate();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(27)->Open( pvSource, pvValue, adOpenDynamic, adLockBatchOptimistic, -1 );
ADO_Recordset(27)->CancelUpdate();
```

ADO_Recordset(n)->Clone

Duplicates an ADO Recordset object. Can specify that the clone be read-only.

Use to create duplicate ADO Recordset objects, especially if you want to maintain more than one current record in a given set of records. Using this method is more efficient than creating and opening a new ADO Recordset object with the same definition as the original.

Syntax

```
ADO_Recordset(n)->Clone( ADOLockTypeEnum nLock, CARecordSet* pRecordSet, CADOLoadBookmark&
cBookmarks );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nLock	*ADOLockTypeEnum* This is the type of locking that should occur to the recordset while the cloning operation is taking place. Valid values are: Value — Description adLockUnspecified — Unspecified lock option adLockReadOnly — Read-only lock adLockPessimistic — Pessimistic lock adLockOptimistic — Optimistic lock adLockBatchOptimistic — Batch optimistic lock
pRecordSet	This is the new instance of the ADO Recordset cloned from the calling ADO_Recordset(n).
cBookmarks	The globally available container of LoadBookmarks.

Example

```
ADO_Recordset(1)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
ADO_Recordset(1)->GetEOF( pVTBOOL );
ADO_Recordset(1)->Clone(adLockOptimistic, ADORecordset[2], ADOBM );
```

ADO_Recordset(n)->Close

Closes an open object and any dependent objects.

When used to close an ADO Recordset, it releases the associated data and any exclusive access you may have had to the data through this object.

ActiveX Data Objects (ADO) comprises a series of objects, which have states. In the ADO Recordset and ADO Connect objects, it is important to close the object before releasing the object.

Syntax

```
ADO_Recordset(n)->Close();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * FROM Test_Table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Close();
ADORecordset.Release( 0, ADOBM );
```

ADO_Recordset(n)->CompareBookmarks

Compares two bookmarks and returns an indication of their relative values.

Compared bookmarks must apply to the same ADO Recordset object or an ADO Recordset object and its clone. Bookmarks from different ADO Recordset objects can't be compared reliably, even when created from the same source or command. An ADO Recordset object's underlying provider must support comparisons.

Syntax

```
ADO_Recordset(n)->CompareBookmarks( CLoadBookmark* pBM1, CLoadBookmark* pBM2, long* pCompare
);
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pBM1	A pointer to a CLoadBookmark.
pBM2	A pointer to a CLoadBookmark.
pCompare	A pointer to a long, containing the return value.

Example

```
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
BeginCheckpoint("ADORecordset::CompareBookmarks");
ADO_Recordset(0)->CompareBookmarks( ADOBM[0], ADOBM[0], pLong );
EndCheckpoint("ADORecordset::CompareBookmarks");
```

ADO_Recordset(n)->Delete

Use to delete the current record or a group of records.

This method marks the current record or a group of records in an ADO Recordset object for deletion. If the object does not allow record deletion, an error occurs. In immediate update mode, deletions occur immediately.

Syntax

```
ADO_Recordset(n)->Delete( ADOAffectEnum nAffect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAffect	ADOAffectEnum The recordset affect enumerator. Valid values include: Value — Description adAffectCurrent — Affects current only adAffectGroup — Affects group adAffectAll — Affects all adAffectAllChapters — Affects all chapters

Example

```
ADO_Recordset(0)->AddNew( pvSource, pvValue );
EndCheckpoint("ADORecordset::AddNew");
BeginCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Find("test_number = 99", 0, adSearchForward, ADOBM[2] );
EndCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Delete( adAffectCurrent );
```

ADO_Recordset(n)->Find

Locates a row in an ADO Recordset that matches specified criteria.

You may specify the search direction, starting row, and offset from the starting row. When the criteria is met, the found record becomes the current row position. If not met, the current row position is set to the end or start of the ADO Recordset.

Syntax

```
ADO_Recordset(n)->Find ( char* sSearchString, long nStartOffset, ADOSearchDirectionEnum
nUpOrDown, CLoadBookmark* pBMStart );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.Criteria String value containing a statement that specifies the column name, comparison operator, and value to use in the search.
sSearchString	String value containing a statement that specifies the column name, comparison operator, and value to use in the search.
nStartOffset	Long value specifying the row offset from the current row or Start bookmark to begin the search. Default is zero. The search starts on the current row, by default.
nUpOrDown	ADOSearchDirectionEnum (Optional) SearchDirectionEnum value specifying if the search should begin on the current or next available row in the direction of the search. Valid values are: Value — Description adSearchForward — Search direction forward adSearchBackward — Search direction backward
pBMStart	(Optional) Variant bookmark that is the starting position for the search.

Example

```
ADO_Recordset(0)->AddNew( pvSource, pvValue );
EndCheckpoint("ADORecordset::AddNew");
BeginCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Find("test_number = 99", 0, adSearchForward, ADOBM[2] );
EndCheckpoint("ADORecordset::Find");
ADO_Recordset(0)->Delete( adAffectCurrent );
```

ADO_Recordset(n)->GetAbsolutePage

Identifies, by page number, where the current record resides.

Syntax

```
ADO_Recordset(n)->GetAbsolutePage( long* pPage );
```

Return Value

Parameters

Parameter	Description

n	An index to the object.
pPage	Pointer to a long returning the page number.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetAbsolutePosition

Specifies the ordinal position of the current record of an ADO Recordset object.

Use this method to locate a record based on its ordinal position, or to determine the current record's ordinal position. This is only available if your provider supports the appropriate functionality.

Syntax

```
ADO_Recordset(n)->GetAbsolutePosition( long* pAbsPos );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pAbsPos	Pointer to a long.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetActiveCommand

Specifies the ADO Command object that created an ADO Recordset object.

A Null object reference is returned if the ADO Recordset was not created by an ADO Command object.

Use this property to determine the ADO Command object when only the ADO Recordset object is known. This function is only converted if there is an ADO Command object associated with this ADO Recordset. This is determined at conversion time.

Syntax

```
ADO_Recordset(n)->GetActiveCommand( IDispatch** ppIDispatch );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
ppIDispatch	Pointer to pointer to IDispatch reference to object.

ADO_Recordset(n)->GetActiveConnection

For a Command, ADO Recordset, or ADO Record object, specifies the associated ADO Connect object.

This property is read-only for open ADO Recordset objects or those whose Source property is set to a valid Command object. Otherwise, it is read/write.

Syntax

```
ADO_Recordset(n)->GetActiveConnection( VARIANT* pvConnection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvConnection	Pointer to the Connection object.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetBOF

Determines if an ADO Recordset object contains records or if you've gone beyond its limits while moving from record to record.

If the current record position is before the first record, GetBOF returns True (-1). If it is on or after the first record, GetBOF returns False (0).

Syntax

```
ADO_Recordset(n)->GetBOF( VARIANT_BOOL* pBOF );
```

Return Value

1 (True) if the current record position is before the first record.

0 (False) if the current is on or after the first record.

Parameters

Parameter	Description
n	An index to the object.
pBOF	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->GetBookmark

Indicates a bookmark identifying an ADO Recordset object's current record, or sets the current record to that identified by a bookmark.

Use to save the position of the current record and return to it at any time. Bookmarks are available only in ADO Recordset objects that support bookmark functionality.

Syntax

```
ADO_Recordset(n)->GetBookmark( CLoadBookmark* pBookmark );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pBookmark	A Pointer to a CLoadBookmark instance.

Example

```
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->GetBookmark( ADOBM[0] );
```

ADO_Recordset(n)->GetCacheSize

Specifies the number of records in the ADO Recordset that are cached locally.

Use to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory.

Syntax

```
ADO_Recordset(n)->GetCacheSize( long* pSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pSize	Pointer to a long value of the number of records in the ADO Recordset cached locally.

Example

```
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 6 );
```

ADO_Recordset(n)->GetCollect

This is a hidden method. It is undocumented within MSDN. If you are looking at incorporating this method, please examine the example below.

Note: Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->GetCollect( VARIANT* pvIndex, VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvIndex	A Pointer to a variant – perhaps the field name or ordinal.
pvData	A Pointer to a variant– perhaps the data for that field.

Example

```
ADO_Recordset(5)->GetState( pLong );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
```

```
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_Recordset(5)->GetCollect( pvValue, pvData );
```

ADO_Recordset(n)->GetCursorLocation

Specifies the library that the cursor service uses.

Allows you to choose between various cursor libraries accessible to the provider. Normally, the library can be client-side or on the server.

Syntax

```
ADO_Recordset(n)->GetCursorLocation( long* pCursorLoc );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCursorLoc	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 1 );
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseClient );
```

ADO_Recordset(n)->GetCursorType

Specifies the type of cursor to use when opening the ADO Recordset object.

If the CursorLocation property is set to adUseClient, the only setting supported is adOpenStatic. If an unsupported value is set, the closest supported CursorType will be used instead.

Syntax

```
ADO_Recordset(n)->GetCursorType( long* pCursorType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pCursorType	A pointer to a long's representation of the CursorLocationEnum returned by the call. This is then sent back to the script for the user.

Example

```
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenDynamic );
```

ADO_Recordset(n)->GetDataMember

Specifies the data member to be retrieved from the object referenced by the DataSource property.

Creates data-bound controls with the Data Environment.

Syntax

```
ADO_Recordset(n)->GetDataMember( CLoadString& sDataMember );
```

Return Value

\

Parameters

Parameter	Description
n	An index to the object.
sDataMember	A CLoadString containing a string representation of the data Member.

Example

```
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenKeyset, adLockOptimistic, -1 );
ADO_Recordset(0)->GetDataMember( sLoadStr );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
```

ADO_Recordset(n)->GetDataSource

Specifies an object containing data to be represented as an ADO Recordset object.

Creates data-bound controls with the Data Environment. GetDataSource takes a handle to an IUnknown as its argument. This is a pointer to a pointer. Please be careful dereferencing this element.

Syntax

```
ADO_Recordset(n)->GetDataSource( IUnknown** ppIUnknown );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
ppIUnknown	A pointer to a pointer to a returned COM object.

Example

```
ADO_Recordset(1)->GetDataSource( &pIUnknown );
ADO_Recordset(2)->GetActiveConnection( pvValue );
ADO_Recordset(2)->GetCursorType( pLong );
```

ADO_Recordset(n)->GetEditMode

Specifies the current record's editing status.

Indicates whether changes have been made to this buffer associated with the current record, or whether a new record has been created. Use to determine the current record's editing status.

Syntax

```
ADO_Recordset(n)->GetEditMode( long* pEMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pEMode	A pointer to a long.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```

ADO_Recordset(n)->GetEOF

Indicates that the current record position is after the last record in an ADO Recordset object.

Syntax

```
ADO_Recordset(n)->GetEOF( VARIANT_BOOL* pEOF );
```

Return Value

1(True) if the current record position is after the last record.

0 (False) if the current record is on or before the last record.

Parameters

Parameter	Description
n	An index to the object.
pEOF	True (-1) or false (0).

Example

```
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->GetFields

Returns a container of an ADO Recordset or ADO Record object's Field objects.

QALoad 's implementation of the GetFields method takes care of making the call through to the GetFields method within the ADO Recordset object. The Argument is one of the ADOFieldSet elements.

Retrieves an ADO Recordset object's ADO FieldSet object. This is an important step in variablization.

Note: This function is not currently being converted in the script; however, this method can be used in conjunction with ADO_Field(n)->GetItem( ) to return data from a specific field of a particular recordset. It can be turned on or off using the QALoad Script Development Workbench's Convert Options wizard.

Syntax

```
ADO_Recordset(n)->GetFields( CAFieldSet* pFieldSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pFieldSet	An instance of the container of fields for a particular recordset.

Example

The following example illustrates returning the first field from the current recordset.

```
ADO_LoadVariant( pvSource, "8", "select * from test_table" );
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;
dsn=" "FhLoadDB2;uid=sa;pwd=;database=Master;" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenDynamic, adLockPessimistic, 1 );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
ADO_FieldSet(0)->GetCount( pLong );
```

```
ADO_FieldSet(0)->Refresh();
ADO_LoadVariant( pvValue, "2", "1" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
```

ADO_Recordset(n)->GetFilter

Specifies a filter for data in an ADO Recordset.

Use to screen out records in an ADO Recordset object. The filtered ADO Recordset becomes the current cursor.

Syntax

```
ADO_Recordset(n)->GetFilter( VARIANT* pvFilter );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvFilter	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```

ADO_Recordset(n)->GetIndex

This is a hidden method. It is undocumented within MSDN.

Note: Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->GetIndex( CLoadString& sIndex );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sIndex	A CLoadString object encapsulating some string data.

ADO_Recordset(n)->GetLockType

Specifies the type of locks placed on records during editing.

Set before opening an ADO Recordset to determine what type of locking the provider should use when opening the ADO Recordset. Read the property to return the type of locking in us.

Syntax

```
ADO_Recordset(n)->GetLockType( long* pLockType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pLockType	A pointer to a long.

Example

```
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenForwardOnly );
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockOptimistic );
```

ADO_Recordset(n)->GetMarshalOptions

Specifies records to be marshaled back to the server.

Syntax

```
ADO_Recordset(n)->GetMarshalOptions( long* pMO );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pMO	Pointer to a long.

Example

```
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
ADO_Recordset(0)->GetMarshalOptions( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
```

ADO_Recordset(n)->GetMaxRecords

Specifies the maximum number of records to return to an ADO Recordset from a query.

Use to limit the number of records that the provider returns. The default, zero, indicates the provider returns all requested records.

Syntax

```
ADO_Recordset(n)->GetMaxRecords( long* pMaxRecs );
```

Return Value

Parameters

Parameters	Description
n	An index to the object.
pMaxRecs	A pointer to a long.

Example

```
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockReadOnly );
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->PutMaxRecords( 10 );
```

ADO_Recordset(n)->GetPageCount

Specifies the number of pages of data contained in the ADO Recordset object.

Syntax

```
ADO_Recordset(n)->GetPageCount( long* pPages );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pPages	A pointer to a long.

Example

```
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageCount( pLong );
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->GetPageSize

Indicates the number of records that make up a single page in the ADO Recordset.

Use to determine how many records make up a logical page of data, which allows you to use the AbsolutePage property.

Syntax

```
ADO_Recordset(n)->GetPageSize( long* pPSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pPSize	A pointer to a long.

Example

```
ADO_Recordset(0)->GetMarshalOptions( pLong );
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageCount( pLong );
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->GetProperties

Retrieves the complete set of properties for this particular instance of the Recordset object.

The CAField object has a collection of property objects. Each property object corresponds to a characteristic of the ADO object specific to the provider. Property sets may change for different providers.

Syntax

```
ADO_Recordset(n)->GetProperties( CAPropertySet* pPropertySet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pPropertySet	Set of CAProperty objects. Each CAProperty object contains a single characteristic, a piece of data, that partially describes the state of a particular instance of an object.

Example

```
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->GetState( pLong );
ADO_Recordset(0)->GetProperties( ADOPropertySet[0] );
ADOPropertySet.Release( 0 );
```

ADO_Recordset(n)->GetRecordCount

Indicates the number of records in an ADO Recordset object.

If ADO cannot determine the number, or if the provider or cursor type doesn't support RecordCount, GetRecordCount returns -1. An error results if GetRecordCount is used on a closed ADO Recordset.

Syntax

```
ADO_Recordset(n)->GetRecordCount( long* pNumRecs );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pNumRecs	A pointer to a long.

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->GetRows

Retrieves multiple records of an ADO Recordset object into an array.

Versions

Versions of ADO_Recordset(n)->GetRows are:

ADO_Recordset(n)->GetRows(long nNumRows, VARIANT* pvStart, VARIANT* pvFields, VARIANT* pvReturnedRows);

ADO_Recordset(n)->GetRows(long nNumRows, CLoadBookmark* pBMStart, VARIANT* pvFields, VARIANT* pvReturnedRows);

ADO_Recordset(n)->GetSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

Syntax

```
ADO_Recordset(n)->GetSort( CLoadString& sSort );
```

Return Value

\

Parameters

Parameter	Description
n	An index to the object.
sSort	A CLoadString containing the field to sort by.

ADO_Recordset(n)->GetSource

Indicates the data source for a Recordset object.

Use the Source property to specify a data source for a Recordset object using one of the following: a Command object variable, an SQL statement, a stored procedure, or a table name. The Variant that is passed into the function is initialized before the call is made so that it properly receives the variant information coming back from the call.

Syntax

```
ADO_Recordset(n)->GetSource( VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvValue	A pointer to a VARIANT.

Example

```
ADO_LoadVariant( pvValue, "3", "0" );
ADO_Recordset(0)->PutFilter( pvValue );
ADO_Recordset(0)->GetSource( pvSource );
ADO_Recordset(0)->GetStatus( pLong );
```

ADO_Recordset(n)->GetState

Indicates for all applicable objects whether the state of the object is open or closed.

Indicates for all applicable objects executing an asynchronous method, whether the current state of the object is connecting, executing, or retrieving.

Syntax

```
ADO_Recordset(n)->GetState( long* pState );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pState	A pointer to a long.

Example

```
ADO_Recordset(0)->PutMaxRecords( 0 );
ADO_Recordset(0)->GetState( pLong );
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
```

ADO_Recordset(n)->GetStatus

Indicates the status of the current record with respect to batch updates or other bulk operations.

Syntax

```
ADO_Recordset(n)->GetStatus( long* pStatus );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pStatus	A pointer to a long.

Example

```
ADO_LoadVariant( pvValue, "3", "0" );
ADO_Recordset(0)->PutFilter( pvValue );
ADO_Recordset(0)->GetSource( pvSource );
ADO_Recordset(0)->GetStatus( pLong );
```

ADO_Recordset(n)->GetStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

This property applies to hierarchical recordsets, such as those supported by the Microsoft Data Shaping Service for OLE DB.It must be set on the parent ADO Recordset before the child ADO Recordset is retrieved. This property simplifies navigating hierarchical recordsets.

Since the VARIANT_BOOL datatype used by ADO is a direct mapping to the short datatype, QALoad uses the short datatype for this call.

Syntax

```
ADO_Recordset(n)->GetStayInSync( short* pStayInSync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pStayInSync	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
ADO_Recordset(0)->PutStayInSync( FALSE );
ADO_LoadVariant( pvSource, "8", "select * from test_table where keyval < 100" ;
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;dsn="
"FhLoadDB2;uid=sa;pwd=;database=Master;" );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenUnspecified, adLockUnspecified, -1 );
EndCheckpoint("ADORecordset::Open");
```

ADO_Recordset(n)->GetString

Returns the ADO Recordset as a string.

Row data, but no schema data, is saved to the string. Therefore, an ADO Recordset cannot be re-opened using this string.

Syntax

```
ADO_Recordset(n)->GetString( ADOStringFormatEnum nStringFormat, long nNumRows, char*
sColDelimitString, char* sRowDelimitString, char* sNullExpString, CLoadString& sRetString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nStringFormat	ADOStringFormatEnum The string format. Valid values are: Value — Description adClipString — Clip String
nNumRows	Number of rows to be saved to string.
sColDelimitString	String with the column delimiter.
sRowDelimitString	String with the row delimiter.
sNullExpString	String with the NULL expression.
sRetString	A CLoadString containing the string being returned.

Example

```
ADO_Recordset(0)->MoveFirst();
BeginCheckpoint("ADORecordset::GetString");
ADO_Recordset(0)->GetString( adClipString, -1, "", "", "", sLoadStr );
EndCheckpoint("ADORecordset::GetString");
ADO_LoadVariant( pvValue, "3", "1" );
BeginCheckpoint("ADORecordset::Move");
ADO_Recordset(0)->Move( 5, pvValue );
EndCheckpoint("ADORecordset::Move");
```

ADO_Recordset(n)->Move

Moves the position of the current record in an ADO Recordset object.

If the NumRecords argument is greater than zero, the current record position moves forward, toward the end of the ADO Recordset. If NumRecords is less than zero, the current record position moves backward, toward the beginning of the ADO Recordset.

If the Move call moves the current record position to a point before the first record, ADO sets the current record to the position before the first record in the recordset (BOF is True). An attempt to move backward when the BOF property is already True generates an error.

Syntax

```
ADO_Recordset(n)->Move( long nNumRecs, VARIANT* pVar );
```

or

```
ADO_Recordset(n)->Move( long nNumRecs, CLoadBookmark* pBM );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nNumRecs	Specifies the number of records that the current record position moves.
pVar	VARIANT pointer to record value.
pBM	Pointer to CLoadBookmark object serving as the starting point.

Example

```
BeginCheckpoint("ADORecordset::GetString");
ADO_Recordset(0)->GetString( adClipString, -1, "", "", "", sLoadStr );
EndCheckpoint("ADORecordset::GetString");
ADO_LoadVariant( pvValue, "3", "1" );
BeginCheckpoint("ADORecordset::Move");
ADO_Recordset(0)->Move( 5, pvValue );
EndCheckpoint("ADORecordset::Move");
```

ADO_Recordset(n)->MoveFirst

Use the MoveFirst method to move the current record position to the first record in the ADO Recordset.

Syntax

```
ADO_Recordset(n)->MoveFirst();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * From test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->MoveLast

Use the MoveLast method to move the current record position to the last record in the ADO Recordset.

The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call generates an error.

Syntax

```
ADO_Recordset(n)->MoveLast();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
```

ADO_Recordset(n)->MoveNext

Use the MoveNext method to move the current record position one record forward, toward the bottom of the ADO Recordset.

If the last record is the current record and you call the MoveNext method, ADO sets the current record to the position after the last record in the ADO Recordset (EOF is True). An attempt to move forward when the EOF property is already True generates an error.

Syntax

```
ADO_Recordset(n)->MoveNext();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * From test_table " );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->MoveFirst();
```

```
ADO_Recordset(0)->GetEOF( pVTBOOL );
ADO_Recordset(0)->MoveNext();
```

ADO_Recordset(n)->MovePrevious

Moves the current record position one record backward, toward the top of the ADO Recordset.

The ADO Recordset object must support bookmarks or backward cursor movement; otherwise, the method call generates an error. If the first record is the current record and you call the MovePrevious method, ADO sets the current record to the position before the first record in the ADO Recordset (BOF is True).

Syntax

```
ADO_Recordset(n)->MovePrevious();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
```

ADO_Recordset(n)->NextRecordset

Clears the current ADO Recordset object and returns the next ADO Recordset by advancing through a series of commands.

Use the NextRecordset method to return the results of the next command in a compound command statement or of a stored procedure that returns multiple results. If you open an ADO Recordset object based on a compound command statement, for example, "SELECT * FROM table1;SELECT * FROM table2" using the Execute method on a Command or the Open method on an ADO Recordset, ADO executes only the first command and returns the results to recordset.

Syntax

```
ADO_Recordset(n)->NextRecordset( VARIANT* pvAffects, CARecordSet* pADORecordSet );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

pvAffects	A pointer to a VARIANT.
pADORecordSet	ADORecordset instantiated by the returned data.

ADO_Recordset(n)->Open

Using the Open method on an ADO Recordset object opens a cursor that represents records from a base table, the results of a query, or a previously saved ADO Recordset.

Syntax

```
ADO_Recordset(n)->Open( VARIANT* pvSource, VARIANT* pvConnect, ADOCursorTypeEnum
nCursorType, ADOLockTypeEnum nLockType, ADOCommandTypeEnum nOptions );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a VARIANT.
pvConnect	A pointer to a VARIANT.
nCursorType	ADOCursorTypeEnum The CursorTypeEnum argument can be: Value — Description adOpenUnspecified — Unspecified Cursor Open option adOpenForwardOnly — Forward-only Cursor adOpenKeyset — KeySet cursor adOpenDynamic — Dynamic cursor adOpenStatic — Static cursor
nLockType	ADOLockTypeEnum This is the type of locking that should occur to the recordset while the cloning operation is taking place. Valid values are: Value — Description adLockUnspecified — Unspecified lock option

	adLockReadOnly	Read-only lock
	adLockPessimistic	Pessimistic lock
	adLockOptimistic	Optimistic lock
	adLockBatchOptimistic	Batch optimistic lock
nOptions	ADOCommandTypeEnum A CommandTypeEnum. Valid values are:	
	Value	Description
	adCmdUnspecified	Does not specify the command type argument
	adCmdUnknown	Default. Indicates that the type of command in the CommandText property is not known
	adCmdText	Evaluates CommandText as a textual definition of a command or stored procedure call
	adCmdTable	Evaluates CommandText as a table name whose columns are all returned by an internally generated SQL query
	adCmdStoredProc	Evaluates CommandText as a stored procedure name
	adCmdFile	Evaluates CommandText as the file name of a persistently stored Recordset
	adCmdTableDirect	Evaluates CommandText as a table name whose columns are all returned

Example

```
ADO_LoadVariant( pvSource, "8", "SELECT * FROM Test_Table" );
LoadVariant( pvValue, ADOConnect[0] );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
EndCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Close();
ADORecordset.Release( 0, ADOBM );
```

ADO_Recordset(n)->PutAbsolutePage

Indicates on which page the current record resides.

Use the AbsolutePage property to identify the page number on which the current record of the ADO Recordset is located.

Syntax

```
ADO_Recordset(n)->PutAbsolutePage( ADOPositionEnum nPage );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nPage	ADOPositionEnum The file position. Valid values are: Value — Description adPosUnknown — Unknown record position adPosBOF — Position at beginning of file adPosEOF — Position at end of file

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->PutAbsolutePosition

Indicates the ordinal position of an ADO Recordset object's current record.

Use the AbsolutePosition property to move to a record based on its ordinal position in the ADO Recordset object, or to determine the ordinal position of the current record. The provider must support the appropriate functionality for this property to be available.

Syntax

```
ADO_Recordset(n)->PutAbsolutePosition( ADOPositionEnum nAbsPos );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAbsPos	ADOPositionEnum The file position. Valid values are:

	Value	Description
	adPosUnknown	Unknown record position
	adPosBOF	Position at beginning of file
	adPosEOF	Position at end of file

Example

```
ADO_Recordset(0)->GetAbsolutePage( pLong );
ADO_Recordset(0)->PutAbsolutePage( (PositionEnum)9 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetAbsolutePosition( pLong );
ADO_Recordset(0)->PutAbsolutePosition( (PositionEnum)38 );
ADO_Recordset(0)->GetActiveConnection( pvValue );
```

ADO_Recordset(n)->PutActiveConnection

Indicates to which Connection object the specified Command, ADO Recordset or Record object, currently belongs.

For open ADO Recordset objects or for ADO Recordset objects whose Source property is set to a valid Command object, the ActiveConnection property is read-only. Otherwise, it is read/write.

Syntax

```
ADO_Recordset(n)->PutActiveConnection( VARIANT* pvConnection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvConnection	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->PutBookmark

Indicates a bookmark that uniquely identifies the current record in an ADO Recordset object or sets the current record in an ADO Recordset object to the record identified by a valid bookmark.

Use the Bookmark property to save the position of the current record and return to that record at any time. Bookmarks are available only in ADO Recordset objects that support bookmark functionality.

Syntax

```
ADO_Recordset(n)->PutBookmark( CLoadBookmark* pBookmark );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pBookmark	A CLoadBookmark object containing a bookmark associated with the element.

Example

```
BeginCheckpoint("ADORecordset::GetBookmark");
ADO_Recordset(0)->GetBookmark( ADOBM[0] );
EndCheckpoint("ADORecordset::GetBookmark");
ADO_Recordset(0)->MoveLast();
BeginCheckpoint("ADORecordset::PutBookmark");
ADO_Recordset(0)->PutBookmark( ADOBM[0] );
EndCheckpoint("ADORecordset::PutBookmark");
```

ADO_Recordset(n)->PutCacheSize

Indicates the number of records in the ADO Recordset that are cached locally.

Use the CacheSize property to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory. For example, if the CacheSize is 10, after first opening the ADO Recordset object, the provider retrieves the first 10 records into local memory.

Syntax

```
ADO_Recordset(n)->PutCacheSize( long nSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nSize	The size of the cache, a positive integer.

Example

```
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
```

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 6 );
```

ADO_Recordset(n)->PutCollect

This is a hidden method. It is undocumented within MSDN. If you want to incorporate this method, examine the example below. Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->PutCollect( VARIANT* pvIndex, VARIANT* pvValue );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvIndex	A Pointer to a variant – perhaps the field name or ordinal.
pvValue	A Pointer to a variant – perhaps the data for that field.

Example

```
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_LoadVariant( pvData, "8", "333555333" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
ADO_LoadVariant( pvData, "8", "Gifford" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
ADO_LoadVariant( pvData, "8", "Roger" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_LoadVariant( pvData, "8", "X" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
```

ADO_Recordset(n)->PutCursorLocation

Indicates the location of the cursor service.

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

Syntax

```
ADO_Recordset(n)->PutCursorLocation( ADOCursorLocationEnum nCursorLoc );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nCursorLoc	ADOCursorLocationEnum The place from which the cursor is drawn. It can be any one of the following: Value — Description adUseNone — No specified cursor location adUseServer — Server-side cursor adUseClient — Client-side cursor adUseClientBatch — Batch client-side cursor

Example

```
ADO_Recordset(0)->GetCacheSize( pLong );
ADO_Recordset(0)->PutCacheSize( 1 );
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseClient );
```

ADO_Recordset(n)->PutCursorType

Specifies the type of cursor to use when opening the ADO Recordset object.

Only a setting of adOpenStatic is supported if the CursorLocation property is set to adUseClient. If an unsupported value is set, then no error results. The closest supported CursorType is used instead.

Syntax

```
ADO_Recordset(n)->PutCursorType( ADOCursorTypeEnum nCursorType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nCursorType	ADOCursorTypeEnum The CursorTypeEnum argument can be: Value — Description adOpenUnspecified — Unspecified Cursor Open option adOpenForwardOnly — Forward-only Cursor

	adOpenKeyset	KeySet cursor
	adOpenDynamic	Dynamic cursor
	adOpenStatic	Static cursor

Example

```
ADO_Recordset(0)->GetCursorLocation( pLong );
ADO_Recordset(0)->PutCursorLocation( adUseServer );
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenDynamic );
```

ADO_Recordset(n)->PutDataMember

Indicates the name of the data member that is retrieved from the object referenced by the DataSource property.

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (data members) that are represented as an ADO Recordset object.

Syntax

```
ADO_Recordset(n)->PutDataMember( char* sDataString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sDataString	Name of the data member being returned from the Recordset object.

ADO_Recordset(n)->PutFilter

Indicates a filter for data in an ADO Recordset.

Use the Filter property to selectively screen out records in an ADO Recordset object. The filtered ADO Recordset becomes the current cursor.

Syntax

```
ADO_Recordset(n)->PutFilter( VARIANT* pvFilter );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvFilter	A pointer to a VARIANT.

Example

```
ADO_Recordset(0)->GetEditMode( pLong );
ADO_Recordset(0)->GetFilter( pvValue );
ADO_LoadVariant( pvValue, "8", "tinyint_col = 99" );
ADO_Recordset(0)->PutFilter( pvValue );
```

ADO_Recordset(n)->PutIndex

This is a hidden method. It is undocumented within MSDN. If you want to incorporate this method, examine the example below. Neither QALoad support professionals nor development recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->PutIndex( char* sIndexString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sIndexString	The Index on the recordset.

ADO_Recordset(n)->PutLockType

Indicates the type of locks placed on records during editing.

Set the LockType property before opening an ADO Recordset to specify what type of locking the provider should use when opening it. Read the property to return the type of locking in use on an open ADO Recordset object.

Note: The LockTypeEnum argument can be any of several elements listed below, or it may be a cast number (LockTypeEnum)0. For best results when load testing, feel free to replace the lock type with adLockOptimistic.

adLockUnspecified
adLockReadOnly
adLockPessimistic

adLockOptimistic
adLockBatchOptimistic

Syntax

```
ADO_Recordset(n)->PutLockType( LockTypeEnum nLockType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nLockType	An enumeration of lock types.

Example

```
ADO_Recordset(0)->GetCursorType( pLong );
ADO_Recordset(0)->PutCursorType( adOpenForwardOnly );
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockOptimistic );
```

ADO_Recordset(n)->PutMarshalOptions

Indicates which records are to be marshaled back to the server.

Syntax

```
ADO_Recordset(n)->PutMarshalOptions( ADOMarshalOptionsEnum nMO );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nMO	ADOMarshalOptionsEnum Indicator about records to send across. valid values are: Value — Description adMarshalAll — Marshal all records adMarshalModifiedOnly — Marshal only modified records

Example

```
ADO_Recordset(2)->PutMarshalOptions( adMarshalModifiedOnly );
ADO_LoadVariant( pvValue, "8", "sSSN" );
ADO_LoadVariant( pvData, "8", "333555333" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sLastName" );
```

```
ADO_LoadVariant( pvData, "8", "Gifford" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sFirstName" );
ADO_LoadVariant( pvData, "8", "Roger" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
ADO_LoadVariant( pvValue, "8", "sMiddleInitial" );
ADO_LoadVariant( pvData, "8", "X" );
ADO_Recordset(2)->PutCollect( pvValue, pvData );
```

ADO_Recordset(n)->PutMaxRecords

Indicates the maximum number of records to return to an ADO Recordset from a query.

Use the MaxRecords property to limit the number of ADO Records that the provider returns from the data source. The default setting of this property is zero, which means the provider returns all requested records.

Syntax

```
ADO_Recordset(n)->PutMaxRecords( long nMaxRecs );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nMaxRecs	Maximum number of records to return from a Recordset query.

Example

```
ADO_Recordset(0)->GetLockType( pLong );
ADO_Recordset(0)->PutLockType( adLockReadOnly );
ADO_Recordset(0)->GetMaxRecords( pLong );
ADO_Recordset(0)->PutMaxRecords( 10 );
```

ADO_Recordset(n)->PutPageSize

Indicates how many records constitute one page in the ADO Recordset.

Use the PageSize property to determine how many ADO Records make up a logical page of data. Establishing a page size allows you to use the AbsolutePage property to move to the first record of a particular page.

Syntax

```
ADO_Recordset(n)->PutPageSize( long nPSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nPSize	Number of records forming a page in the Recordset.

Example

```
ADO_Recordset(0)->GetMarshalOptions( pLong );
ADO_Recordset(0)->GetPageSize( pLong );
ADO_Recordset(0)->PutPageSize( 4 );
ADO_Recordset(0)->GetPageCount( pLong );
ADO_Recordset(0)->GetAbsolutePage( pLong );
```

ADO_Recordset(n)->PutRefActiveConnection

Indicates to which ADO Connect object the specified ADO Command, ADO Recordset, or Record object, currently belongs.

For open ADO Recordset objects or for ADO Recordset objects whose Source property is set to a valid ADO Command object, the ActiveConnection property is read-only. Otherwise, it is read/write.

Syntax

```
ADO_Recordset(n)->PutRefActiveConnection( VARIANT* pvConnection );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvConnection	The connection that is active.

Example

```
ADO_Recordset(0)->GetActiveConnection( pvValue );
ADO_Recordset(0)->GetBOF( pVTBOOL );
ADO_Recordset(0)->PutSource("Select * from test_table where " "keyval < 100" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->PutActiveConnection( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenStatic, adLockPessimistic, -1 );
```

ADO_Recordset(n)->PutRefDataSource

Indicates an object that contains data to be represented as an ADO Recordset object.

This property is used to create data-bound controls with the Data Environment. The Data Environment maintains collections of data (data sources) containing named objects (data members) that will be represented as an ADO Recordset object.

Syntax

```
ADO_Recordset(n)->PutRefDataSource( IUnknown* pIUnknown );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pIUnknown	A pointer to a COM object.

ADO_Recordset(n)->PutRefSource

Sets a Command object as the data source for a Recordset object.

Use the Source property to specify a data source for a Recordset object using one of the following: a Command object variable, SQL statement, stored procedure, or table name.

Syntax

```
ADO_Recordset(n)->PutRefSource( VARIANT* pvSource );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvSource	A pointer to a variant that contains a reference to a valid Command object.

Example

```
ADO_Recordset(0)->PutActiveConnection( pvValue );
LoadVariant( pvValue, ADOCommand[0] );
ADO_Recordset(0)->PutRefSource( pvValue );
ADO_LoadVariant( pvSource, "10", "2147614724" );
```

ADO_Recordset(n)->PutSort

Indicates one or more field names on which the ADO Recordset is sorted, and whether each field is sorted in ascending or descending order.

Syntax

```
ADO_Recordset(n)->PutSort( char* sSortString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSortString	char*

ADO_Recordset(n)->PutSource

Indicates the data source for an ADO Recordset object.

Use the Source property to specify a data source for an ADO Recordset object using one of the following: a Command object variable, an SQL statement, a stored procedure, or a table name.

Syntax

```
ADO_Recordset(n)->PutSource( char* sSourceString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sSourceString	A char* representation of a data source.

Example

```
ADO_Recordset(0)->PutSource( "Select sUID, sPWD, sPhone" "from USER Where lcase(sUID)='sa'
and sPWD = 'sa'" );
ADO_LoadVariant( pvSource, "8", "Select sUID, sPWD, sPhone" "from USER Where
lcase(sUID)='sa' and sPWD = 'sa'" );
LoadVariant( pvValue, ADOConnect[1] );
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenKeyset, adLockOptimistic, -1 );
ADO_Recordset(0)->GetRecordCount( pLong );
ADO_Recordset(0)->GetFields( ADOFieldSet[0] );
```

ADO_Recordset(n)->PutStayInSync

Indicates, in a hierarchical ADO Recordset object, whether the reference to the underlying child records (that is, the chapter) changes when the parent row position changes.

This property applies to hierarchical ADO Recordsets, such as those supported by the Microsoft Data Shaping Service for OLE DB. It must be set on the parent ADO Recordset before the child ADO Recordset is retrieved. This property simplifies navigating hierarchical ADO Recordsets.

Syntax

```
ADO_Recordset(n)->PutStayInSync( short nStayInSync );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nStayInSync	TRUE or FALSE.

Example

```
ADO_Recordset(0)->GetStayInSync( FALSE );
ADO_Recordset(0)->PutStayInSync( pVTBOOL );
ADO_LoadVariant( pvSource, "8", "select * from test_table where keyval < 100" ;
ADO_LoadVariant( pvValue, "8", "PROVIDER=MSDASQL;dsn="
"FhLoadDB2;uid=sa;pwd=;database=Master;" );
BeginCheckpoint("ADORecordset::Open");
ADO_Recordset(0)->Open( pvSource, pvValue, adOpenUnspecified, adLockUnspecified, -1 );
EndCheckpoint("ADORecordset::Open");
```

ADO_Recordset(n)->Requery

Updates the data in an ADO Recordset object by re-executing the query on which the object is based.

Use the Requery method to refresh the entire contents of an ADO Recordset object from the data source by reissuing the original command and retrieving the data a second time.

Syntax

```
ADO_Recordset(n)->Requery( long nOptions );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nOptions	-1 TRUE or 0 FALSE

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
ADO_Recordset(0)->Close();
```

ADO_Recordset(n)->Resync

Refreshes the data in the current ADO Recordset object, or Fields collection of a Record object, from the underlying database.

Use the Resync method to resynchronize records in the current ADO Recordset with the underlying database. This is useful if you are using either a static or forward-only cursor, but you want to see any changes in the underlying database.

Syntax

```
ADO_Recordset(n)->Resync( ADOAffectEnum nAffect, ADOResyncEnum nResyncVals );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAffect	ADOAffectEnum The recordset affect enumerator. Valid values include: Value — Description adAffectCurrent — Affects current only adAffectGroup — Affects group adAffectAll — Affects all adAffectAllChapters — Affects all chapters
nResyncVals	ADOResyncEnum Resync option. Valid values are: Value — Description adResyncUnderlyingValues — Resync the underlying values only adResyncAllValues — Resync all values

Example

```
ADO_Recordset(0)->Resync(adAffectAll, adResyncAllValues );
ADO_Recordset(0)->GetStayInSync( pVTBOOL );
```

ADO_Recordset(n)->Save

Saves the ADO Recordset in a file or ADO Stream object.

The Save method can only be invoked on an open ADO Recordset. Use the Open method to later restore the ADO Recordset from Destination.

If the Filter property is in effect for the ADO Recordset, then only the rows accessible under the filter are saved. If the ADO Recordset is hierarchical, then the current child ADO Recordset and its children are saved, including the parent ADO Recordset. If the Save method of a child ADO Recordset is called, the child and all its children are saved, but the parent is not.

The first time you save the ADO Recordset, it is optional to specify Destination. If you omit Destination, a new file is created with a name set to the value of the Source property of the ADO Recordset.

Note: The second argument here can be given as adPersistXML or adPersistADTG.

Syntax

```
ADO_Recordset(n)->Save( VARIANT* pvDestination, ADOPersistFormatEnum nPersistEnum );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvDestination	A pointer to a VARIANT.
nPersistEnum	ADOPersistFormatEnum The save format. Valid values are: Value — Description adPersistADTG — Persist in ADTG format adPersistXML — Persist in XML format

Example

```
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_LoadVariant( pvValue, "8", "saver.xml" );
ADO_Recordset(0)->Save( pvValue, adPersistXML );
```

ADO_Recordset(n)->Seek

The SeekEnum is an Enumerated value giving the direction of the seek operation.

Syntax

```
ADO_Recordset(n)->Seek( VARIANT* pvKeyValue, ADOSeekEnum nSeekOptions );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvKeyValue	A pointer to a Variant containing an array of Variant values.
nSeekOptions	ADOSeekEnum Direction of the seek. Valid values are: Value — Description adSeekFirstEQ — Seek first equal record adSeekLastEQ — Seek last equal record adSeekAfterEQ — Seek record after found equal record adSeekAfter — Seek after record adSeekBeforeEQ — Seek before found equal record adSeekBefore — Seek before record

ADO_Recordset(n)->Supports

Determines whether a specified ADO Recordset object supports a particular type of functionality.

If the ADO Recordset object supports the features whose corresponding constants are in CursorOptions, the Supports method returns True. Otherwise, it returns False.

Syntax

```
ADO_Recordset(n)->Supports( ADOCursorOptionEnum nCursorOp, VARIANT_BOOL* pBool );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

nCursorOp	ADOCursorOptionEnum Valid cursor type options are: Value — Description adHoldRecords — Hold Records adMovePrevious — Move to previous record adAddNew — Add new record adDelete — Delete record adUpdate — Update record adBookmark — Bookmark record adApproxPosition — Get approximate position adUpdateBatch — Update batch adResync — Resynchonize recordset adNotify — Notify adFind — Find record adSeek — Seek record adIndex — Index of record
pBool	A pointer to a VARIANT_BOOL.

Example

```
ADO_Recordset(0)->MoveFirst();
ADO_Recordset(0)->MoveNext();
ADO_Recordset(0)->MovePrevious();
ADO_Recordset(0)->MoveLast();
ADO_Recordset(0)->Requery( -1 );
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
ADO_Recordset(0)->Close();
```

ADO_Recordset(n)->Update

Saves any changes you make to the current row of an ADO Recordset object.

Use the Update method to save any changes you make to the current record of an ADO Recordset object since calling the AddNew method or since changing any field values in an existing record. The ADO Recordset object must support updates.

Syntax

```
ADO_Recordset(n)->Update( VARIANT* pvFields, VARIANT* pvValues );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvFields	VARIANT pointer to the field(s) to be updated.
pvValues	VARIANT pointer to the value(s) to be updated.

Example

```
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "FirstName" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "3", "John" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_Recordset(2)->GetFields( ADOFieldSet[0] );
ADO_LoadVariant( pvValue, "8", "LastName" );
ADO_FieldSet(0)->GetItem( pvValue, ADOField[0] );
ADO_LoadVariant( pvValue, "8", "Doe" );
ADO_Field(0)->PutValue( pvValue );
ADOFieldSet.Release( 0 );
ADOField.Release( 0 );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
ADO_Recordset(2)->Update( pvValue, pvData );
ADO_Recordset(2)->Close();
```

ADO_Recordset(n)->UpdateBatch

Writes all pending batch updates within the ADO Recordset to disk.

Syntax

```
ADO_Recordset(n)->UpdateBatch( ADOAffectEnum nAffect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAffect	ADOAffectEnum The recordset affect enumerator. Valid values include: Value — Description adAffectCurrent — Affects current only

	adAffectGroup	Affects group
	adAffectAll	Affects all
	adAffectAllChapters	Affects all chapters

Example

```
ADO_Recordset(0)->Delete( adAffectCurrent );
ADO_LoadVariant( pvValue, "10", "2147614724" );
ADO_LoadVariant( pvData, "10", "2147614724" );
BeginCheckpoint("ADORecordset::Update");
ADO_Recordset(0)->Update( pvValue, pvData );
EndCheckpoint("ADORecordset::Update");
BeginCheckpoint("ADORecordset::UpdateBatch");
ADO_Recordset(0)->UpdateBatch( adAffectAll );
EndCheckpoint("ADORecordset::UpdateBatch");
ADO_Recordset(0)->Supports( (CursorOptionEnum)8388608, pVTBOOL );
```

ADO_Recordset(n)->_xClone

This is a hidden method. It is undocumented within MSDN. A logical assumption is that it makes a clone of the calling ADO Recordset. This is given the arguments and the method name.

Note: Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xClone( CARecordSet* pRecSet, CADOLoadBookmark& cBookmarks );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pRecSet	This is the new instance of the ADO Recordset cloned from the calling of this method.
cBookmarks	The global container of ADO Bookmarks.

Example

```
ADO_Recordset(1)->Open( pvSource, pvValue, adOpenStatic, adLockOptimistic, -1 );
ADO_Recordset(1)->GetEOF( pVTBOOL );
ADO_Recordset(1)->_xClone( ADORecordset[2], ADOBM );
```

ADO_Recordset(n)->_xResync

This is a hidden method. It is undocumented within MSDN. A logical assumption is that it re-synchronizes the ADO Recordset with the underlying data provider. This is given the arguments and the method name.

Note: Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xResync( ADOAffectEnum nAffect );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nAffect	ADOAffectEnum The recordset affect enumerator. Valid values include: Value — Description adAffectCurrent — Affects current only adAffectGroup — Affects group adAffectAll — Affects all adAffectAllChapters — Affects all chapters

Example

```
ADO_Connect(1)->Execute( "DELETE FROM MyTemp", pvValue, -1, ADORecordset[4] );
ADO_Recordset(4)->_xResync( adAffectAll );
```

ADO_Recordset(n)->_xSave

This is a hidden method. It is undocumented within MSDN. A logical assumption is that it saves ADO Recordset data to the location given in the first argument. This is given the arguments and the method name.

Note: Compuware does not recommend adding this method to a script.

Syntax

```
ADO_Recordset(n)->_xSave( char* sFileNameString, ADOPersistFormatEnum nPersistEnum );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sFileNameString	The file to which the ADO Recordset information is being saved.
nPersistEnum	ADOPersistFormatEnum

	The save format. Valid values are: Value — Description adPersistADTG — Persist in ADTG format adPersistXML — Persist in XML format

ADO_Stream(n)->Cancel

Cancels execution of a pending ADO Stream, asynchronous method call.

Syntax

```
ADO_Stream(n)->Cancel();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open(pvSource, adModeUnknown, adOpenStreamUnspecified,"","" );
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->Close

Closes an open object and any dependent objects.

Using the Close method to close an ADO Stream object releases the associated data and any exclusive access you may have had to the data through this particular object. You can later call the Open method to reopen the object with the same, or modified, attributes.

Close the ADO Stream and give up all rights you may have had to the data.

Syntax

```
ADO_Stream(n)->Close();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "","" );
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->CopyTo

Copies the specified number of characters or bytes, depending on Type, in the ADO Stream to another ADO Stream object.

This method copies the specified number of characters or bytes, starting from the current position specified by the Position property. If the specified number is more than the available number of bytes until EOS, then only characters or bytes from the current position to EOS are copied. If the value of NumChars is 1, or omitted, all characters or bytes starting from the current position are copied.

If there are existing characters or bytes in the destination ADO Stream, all contents beyond the point where the copy ends remain, and are not truncated. Position becomes the byte immediately following the last byte copied. If you want to truncate these bytes, call SetEOS.

Syntax

```
ADO_Stream(n)->CopyTo( CAStream* pDestStream, long nNumBytes );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pDestStream	An instance of an ADO Source object.
nNumBytes	A positive integer.

Example

```
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "","" );
ADO_Stream(0)->CopyTo( ADOStream[1], 5 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
```

ADO_Stream(n)->Flush

Forces the contents of the ADO Stream remaining in the ADO buffer to the underlying object with which the ADO Stream is associated.

This method may be used to send the contents of the ADO Stream buffer to the underlying object represented by the URL that is the source of the ADO Stream object. This method should be called when you want to ensure that all changes made to the contents of an ADO Stream have been written. However, with ADO it is not usually necessary to call Flush, as ADO continuously flushes its buffer as much as possible in the background.

Changes to the content of an ADO Stream are made automatically, and not cached until Flush is called.

Syntax

```
ADO_Stream(n)->Flush();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(0)->CopyTo( ADOStream[1], 5 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->Flush();
ADO_Stream(0)->Cancel();
ADO_Stream(0)->Close();
```

ADO_Stream(n)->GetCharset

Indicates the character set into which the contents of a text ADO Stream should be translated.

In a text ADO Stream object, text data is stored as Unicode. The Charset property translates the data read from the ADO Stream into the specified character set. Similarly, data written to the ADO Stream in the specified character set is translated into Unicode for storage in the ADO Stream object.

Syntax

```
ADO_Stream(n)->GetCharset( CLoadString& sCharset );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sCharset	A CLoadString object.

Example

```
ADO_Stream(1)->Cancel();
ADO_Stream(1)->Close();
```

```
ADO_Stream(0)->GetCharset( sLoadStr );
ADO_Stream(0)->PutCharset( "Unicode" );
```

ADO_Stream(n)->GetEOS

Indicates whether the current position is at the end of the ADO Stream.

Returns a Boolean value that indicates whether the current position is at the end of the ADO Stream. EOS returns At replay, QALoad checks the current position in the stream to determine whether or not this is the end of the stream.

Syntax

```
ADO_Stream(n)->GetEOS( VARIANT_BOOL* pEOS );
```

Return Value

Bool

True if there are no more bytes in the ADO stream.

False if there are more bytes in the ADO stream following the current position.

Parameters

Parameter	Description
n	An index to the object.
pEOS	A pointer to a VARIANT_BOOL.

Example

```
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->GetLineSeparator

Indicates the binary character to be used as the line separator in text ADO Stream objects.

LineSeparator is used only with text ADO Stream objects (Type is adTypeText). This property is ignored if Type is adTypeBinary.

Syntax

```
ADO_Stream(n)->GetLineSeparator( long* pLineSeparator );
```

Return Value

Parameters

Parameter	Description

n	An index to the object.
pLineSeparator	A pointer to a 4-byte integer.

Example

```
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
ADO_Stream(1)->GetLineSeparator( pLong );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->GetMode

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

Use the Mode property to set or return the access permissions in use by the provider on the current connection. You can set the Mode property only when the Connection object is closed.

Syntax

```
ADO_Stream(n)->GetMode( long* pMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pMode	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetLineSeparator( pLong );
ADO_Stream(0)->PutLineSeparator( adCRLF );
ADO_Stream(0)->GetState( pLong );
ADO_Stream(0)->GetMode( pLong );
ADO_Stream(0)->PutMode( adModeShareDenyNone );
```

ADO_Stream(n)->GetPosition

Indicates the current position within an ADO Stream object.

Sets or returns a Long value that specifies the offset, in number of bytes, of the current position from the beginning of the ADO Stream. The default is 0, which represents the first byte in the ADO Stream.

At replay, QALoad , checks the current position and feeds that position back to the user in the pointer.

Syntax

```
ADO_Stream(n)->GetPosition( long* pPosition );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pPosition	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
```

ADO_Stream(n)->GetSize

Returns a Long value that specifies the size of the ADO Stream in number of bytes.

The default value is the size of the ADO Stream, or -1 if the size of the ADO Stream is not known. At replay, QALoad checks the size of the ADO Stream that is referenced by this call.

Syntax

```
ADO_Stream(n)->GetSize( long* pSize );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pSize	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->PutPosition( 0 );
```

ADO_Stream(n)->GetState

Indicates the state of the ADO Stream object.

The ADO Stream object's State property can have a combination of values. For example, if a statement is executing, this property has a combined value of adStateOpen and adStateExecuting.

GetState returns 0 for a not open state and 1 for an open state.

Syntax

```
ADO_Stream(n)->GetState( long* pState );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pState	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```

ADO_Stream(n)->GetType

Indicates the type of data contained in the ADO Stream, binary or text.

Sets or returns a StreamTypeEnum value that specifies the type of data contained in the ADO Stream object. The default value is adTypeText. However, if binary data is initially written to a new, empty ADO Stream, the Type ischanged to adTypeBinary.

Syntax

```
ADO_Stream(n)->GetType( long* pType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pType	A pointer to a 4-byte integer.

Example

```
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)->GetType( pLong );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
```

ADO_Stream(n)->LoadFromFile

Loads the contents of an existing file into an ADO Stream.

This method may be used to load the contents of a local file into an ADO Stream object. This may be used to upload the contents of a local file to a server.

The ADO Stream object must already be open before calling LoadFromFile. This method does not change the binding of the ADO Stream object. It is still bound to the object specified by the URL with which the ADO Stream was originally opened. LoadFromFile overwrites the current contents of the ADO Stream object with data read from the file.

Syntax

```
ADO_Stream(n)->LoadFromFile( char* sFileNameString );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sFileNameString	A string representation of a file name.

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```

ADO_Stream(n)->Open

Opens an ADO Stream object to manipulate streams of binary or text data.

When a Record object is passed in as the source parameter, the User ID and Password parameters are not used because access to the Record object is already available. Similarly, the Mode of the Record object is transferred to the ADO Stream object.

When Source is not specified, the ADO Stream opened contains no data and has a Size of zero (0). To avoid losing any data that is written to this ADO Stream when the ADO Stream is closed, save the ADO Stream with the CopyTo or SaveToFile methods, or save it to another memory location.

While the ADO Stream is not open, it is possible to read all the read-only properties of the ADO Stream. If an ADO Stream is opened asynchronously, all subsequent operations (other than checking the State and other read-only properties) are blocked until the Open operation is completed.

Open the ADO Stream to manipulate binary or text data.

Syntax

```
ADO_Stream(n)->Open( VARIANT* pvSource, ADOConnectModeEnum nMode, ADOStreamOpenOptionsEnum
nOptions, char* sUserName, char* sPassword );
```

Return Value

Parameters

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>n</td><td>An index to the object.</td></tr>
<tr><td>pvSource</td><td>A pointer to a VARIANT.</td></tr>
<tr><td>nMode</td><td>ADOConnectModeEnum
A ConnectModeEnum value, whose default value is adModeUnknown, that specifies the access mode for the resultant Record object. Valid values are:
Value — Description
adModeUnknown — Unknown connection mode
adModeRead — Read-only mode
adModeWrite — Write-only mode
adModeReadWrite — Read-write mode
adModeShareDenyRead — Exclusive read mode
adModeShareDenyWrite — Exclusive write mode
adModeShareExclusive — Exclusive read-write mode
adModeShareDenyNone — Non-exclusive mode
adModeRecursive — Recursive mode</td></tr>
<tr><td>nOptions</td><td>ADOStreamOpenOptionsEnum
Stream open options. Valid values are:
Value — Description
adOpenStreamUnspecified — Unspecified stream open options
adOpenStreamAsync — Stream opened asynchronously
adOpenStreamFromRecord — Steam opened from record</td></tr>
<tr><td>sUserName</td><td>A user name string.</td></tr>
<tr><td>sPassword</td><td>A password string.</td></tr>
</table>

Example

```
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
```

ADO_Stream(n)->PutCharset

Indicates the character set into which the contents of a text ADO Stream should be translated.

In a text ADO Stream object, text data is stored as Unicode. The Charset property translates the data read from the ADO Stream into the specified character set. Similarly, data written to the ADO Stream in the specified character set is translated into Unicode for storage in the ADO Stream object.

Syntax

```
ADO_Stream(n)->PutCharset( char* sCharset );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sCharset	A string representation of a character set.

Example

```
ADO_Stream(0)->PutCharset( "ascii" );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState(pLong );
ADO_Stream(0)->PutPosition( 15 );
ADO_Stream(0)->GetPosition( pLong );
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)-GetType( pLong );
```

ADO_Stream(n)->PutLineSeparator

Indicates the binary character to be used as the line separator in text ADO Stream objects.

LineSeparator is used only with text ADO Stream objects (Type is adTypeText). This property is ignored if Type is adTypeBinary.

Syntax

```
ADO_Stream(n)->PutLineSeparator( ADOLineSeparatorEnum nLineSeparator );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

nLineSeparator	*ADOLineSeparatorEnum* Line separator options. Valid values are: Value — Description adLF — Linefeed adCR — Carriage return adCRLF — Carriage return and line feed

Example

```
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
```

ADO_Stream(n)->PutMode

Indicates the available permissions for modifying data in a Connection, Record, or ADO Stream object.

Use the Mode property to set or return the access permissions in use by the provider on the current connection.

Note: You can set the Mode property only when the Connection object is closed.

Syntax

```
ADO_Stream(n)->PutMode( ADOConnectModeEnum nMode );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nMode	ADOConnectModeEnum A ConnectModeEnum value, whose default value is adModeUnknown, that specifies the access mode for the resultant Record object. Valid values are: Value — Description adModeUnknown — Unknown connection mode adModeRead — Read-only mode adModeWrite — Write-only mode adModeReadWrite — Read-write mode

	adModeShareDenyRead	Exclusive read mode
	adModeShareDenyWrite	Exclusive write mode
	adModeShareExclusive	Exclusive read-write mode
	adModeShareDenyNone	Non-exclusive mode
	adModeRecursive	Recursive mode

Example

```
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
ADO_Stream(0)->GetSize( pLong );
ADO_Stream(0)->GetState( pLong );
ADO_Stream(0)->PutMode( adModeShareDenyNone );
ADO_Stream(0)->PutPosition( 15 );
```

ADO_Stream(n)->PutPosition

Indicates the current position within an ADO Stream object.

Sets or returns a Long value that specifies the offset, in number of bytes, of the current position from the beginning of the ADO Stream. The default is 0, which represents the first byte in the ADO Stream. At replay, QALoad sets the current position in the ADO Stream.

Syntax

```
ADO_Stream(n)->PutPosition( long nPosition );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nPosition	The ADOStream position

Example

```
ADO_Stream(0)->PutPosition( 0 );
ADO_Stream(0)->GetType( pLong );
ADO_LoadVariant( pvSource, "10", "2147614724" );
ADO_Stream(1)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
```

ADO_Stream(n)->PutType

Indicates the type of data contained in the ADO Stream, binary or text.

Sets or returns a StreamTypeEnum value that specifies the type of data contained in the ADO Stream object. The default value is adTypeText. However, if binary data is initially written to a new, empty ADO Stream, the Type is changed to adTypeBinary.

Syntax

```
ADO_Stream(n)->PutType( ADOStreamTypeEnum nStreamType );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nStreamType	ADOStreamTypeEnum Stream options. Valid values are: Value — Description adTypeBinary — Binary stream adTypeText — Text stream

Example

```
ADO_Stream(0)->Open( pvSource, adModeUnknown, adOpenStreamUnspecified, "", "" );
ADO_Stream(0)->PutType( adTypeText );
ADO_Stream(0)->PutCharset( "ascii" );
ADO_Stream(0)->LoadFromFile( "D:\\Ward.txt" );
```

ADO_Stream(n)->Read

Reads a specified number of bytes from a binary ADO Stream object.

If NumBytes is more than the number of bytes left in the ADO Stream, only the bytes remaining are returned. The data read is not padded to match the length specified by NumBytes. If there are no bytes left to read, a variant with a null value is returned. Read cannot be used to read backwards.

Syntax

```
ADO_Stream(n)->Read( long nNumBytes, VARIANT* pvBuffer );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nNumBytes	A non negative integer value corresponding to the number of bytes to read.

pvBuffer	A pointer to a VARIANT buffer into which the data is read.

ADO_Stream(n)->ReadText

Reads specified number of characters from a text ADO Stream object.

If NumChar is more than the number of characters left in the ADO Stream, only the characters remaining are returned. The string read is not padded to match the length specified by NumChar. If there are no characters left to read, a variant whose value is null is returned.

ReadText cannot be used to read backwards.

Syntax

```
ADO_Stream(n)->ReadText( long nNumChars, CLoadString& sDataRead );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
nNumChars	A non negative integer value corresponding to the number of characters to read.
sDataRead	A CLoadString into which the data is read.

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
```

ADO_Stream(n)->SaveToFile

Saves the number of bytes contents of the current ADO Stream to the file from the current position. It sends the second param number of bytes to that File.

SaveToFile may be used to copy the contents of an ADO Stream object to a local file. There is no change in the contents or properties of the ADO Stream object. The ADO Stream object must be open before calling SaveToFile.

This method does not change the association of the ADO Stream object to its underlying source. The ADO Stream object is still associated with the original URL that was its source when opened.

Syntax

```
ADO_Stream(n)->SaveToFile( char* sFileNameString, ADOSaveOptionsEnum nOptions );
```

Parameters

Parameter	Description
n	An index to the object.
sFileNameString	A String forming the name of the file to save the stream to.
nOptions	*ADOSaveOptionsEnum* Stream write options. Valid values are: Value — Description adSaveCreateNotExist — Save and create if non-existent adSaveCreateOverWrite — Save and overwrite if exists

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 20 );
ADO_Stream(0)->ReadText( 20, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
ADO_Stream(1)->PutLineSeparator( adCR );
ADO_Stream(1)->GetLineSeparator( pLong );
ADO_Stream(1)->SaveToFile( "D:\\StreamReceive.txt", adSaveCreateOverWrite );
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->SetEOS

Sets the current position within the ADO Stream as the End of the ADO Stream.

SetEOS updates the value of the EOS propert, by making the current Position the end of the ADO Stream. Any bytes or characters following the current position are truncated.

Syntax

```
ADO_Stream(n)->SetEOS();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(1)->SaveToFile( "D:\\Streamward.txt", adSaveCreateOverWrite );
ADO_Stream(1)->GetPosition( pLong );
if( *pLong >=20 )
ADO_Stream(1)->SetEOS();
ADO_Stream(1)->Flush();
```

ADO_Stream(n)->SkipLine

Skips one entire line when reading a text ADO Stream.

All characters up to and including the next line separator are skipped. By default, the LineSeparator is adCRLF. If you attempt to skip past EOS, the current position simply remains at EOS.

Skip a line in the text buffer that is the ADO Stream.

Syntax

```
ADO_Stream(n)->SkipLine();
```

Return Value

Parameters

Parameter	Description
n	An index to the object.

Example

```
ADO_Stream(1)->PutType( adTypeText );
ADO_Stream(0)->CopyTo( ADOStream[1], 150 );
ADO_Stream(0)->ReadText( 50, sLoadStr );
ADO_Stream(0)->SkipLine();
ADO_Stream(1)->ReadText( 50, sLoadStr );
ADO_Stream(1)->GetEOS( pVTBOOL );
```

ADO_Stream(n)->Write

Writes BINARY Data to the ADO Stream buffer.

Specified bytes are written to the ADO Stream object without any intervening spaces between each byte. The current Position is set to the byte following the written data. The Write method does not truncate the rest of the data in a stream. If you want to truncate these bytes, call SetEOS.

If you write past the current EOS position, the Size of the ADO Stream is increased to contain any new bytes, and EOS moves to the new last byte in the ADO Stream.

Syntax

```
ADO_Stream(n)->Write( VARIANT* pvWriteBuffer );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
pvWriteBuffer	Pointer to the binary data buffer.

ADO_Stream(n)->WriteText

Writes a specified text string to an ADO Stream object.

Specified strings are written to the ADO Stream object without any intervening spaces or characters between each string.

The current Position is set to the character following the written data. The WriteText method does not truncate the rest of the data in a stream. If you want to truncate these characters, call SetEOS.

Syntax

```
ADO_Stream(n)->WriteText( char* sTextToWrite, ADOStreamWriteEnum nStreamWriteOptions );
```

Return Value

Parameters

Parameter	Description
n	An index to the object.
sTextToWrite	The text to write.
nStreamWriteOptions	ADOStreamWriteEnum Stream write options. Valid values are: Value — Description adWriteChar — Write char adWriteLine — Write line stWriteChar — Stream write character stWriteLine — Stream write line

Example

```
ADO_Stream(1)->GetSize( pLong );
ADO_Stream(1)->PutPosition( *pLong );
ADO_Stream(1)->SetEOS();
ADO_Stream(1)->WriteText( "This is the way we drink water", adWriteLine );
```

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.

CtxConnect
Connects to the Citrix server with the specified hostname and output mode.

CtxDisconnect
Disconnects from the Citrix server.

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

CtxDoubleClick
Double-clicks the mouse at the current location.

CtxKeyDown
Inputs the keystroke specified by the key argument, which corresponds to the VK code.

CtxKeyUp
Inputs the keystroke specified by the key argument, which corresponds to the VK code.

CtxMouseDown
Presses the specified mouse button.

CtxMouseMove
Moves the mouse to the given coordinates with the given button and modifier.

CtxMouseUp
Releases the specified mouse button.

CtxPing
Sends a ping request and waits for the response.

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

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

CtxSetApplication
Connects to the Citrix server with the specified application name and application working directory name.

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

CtxSetConnectTimeout
Sets the number of seconds to wait for connections to the server to complete.

CtxSetDisconnectTimeout
Sets the number of seconds to wait for disconnections from the server to complete.

CtxSetEnableCounters
Enables or disables custom counters for Citrix client-side statistics.

CtxSetEnableWildcardMatching
Enables or disables wildcard and substring name comparisons for matching Citrix window creation events.

CtxSetICAFile
Uses the specified ICA file when connecting to a published application or desktop.

CtxSetLoginInfo
Connects to the Citrix server with the specified user name and password.

CtxSetPingTimeout
Sets the number of seconds to wait for a ping to be acknowledged.

CtxSetWaitPointTimeout
Sets the number of seconds to wait for a wait point.

CtxSetWindowMatchTitle
Sets the string to match the names of previously-created windows.

CtxSetWindowRetries
Sets the retry information for window verification.

CtxSetWindowTimeout
Sets the number of seconds to wait for windows to be activated and destroyed.

CtxSetWindowVerification
Enables or disables window verification for actions.

CtxType
Inputs the specified key strokes.

CtxTypeChar
Sends the specified ASCII character to the Citrix server.

CtxTypeVK
Sends the VK code that corresponds to a key typed by the user.

CtxWaitForCaptionChange
Waits for the specified window's caption to be changed.

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

CtxWaitForWindowActive
Waits for the specified window to be activated (brought to the foreground).

CtxWaitForWindowCreate
Waits for the specified window to be created.

CtxWaitForWindowDestroy
Waits for the specified window to be destroyed.

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();
```

CtxClick

Clicks the specified button, using the specified modifier, at the current location.

Syntax

```
void CtxClick(const CtxWI* windowInfo, long holdTime, CtxMouseButtonEnum button, CtxModifierKeyEnum ModifierKeys);
```

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 mouse button was specified

	L_BUTTON	The left mouse button
	R_BUTTON	The right mouse button
	M_BUTTON	The middle mouse button
ModifierKeys	CtxModifierKeyEnum 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);
```

CtxConnect

Connects to the Citrix server with the specified hostname and output mode.

Syntax

void CtxConnect (const char* hostname, CtxConnectModeEnum outputmode);

Return Value

Parameters

Parameter	Description	
hostname	Name of the server to connect to.	
outputmode	CtxConnectModeEnum Type of playback output/display. The following modes are available:	
	Value	Description
	OUTPUT_MODE_NORMAL	Seamless rendering and display with

		window management.
	OUTPUT_MODE_WINDOWLESS	Graphics are not displayed.
	OUTPUT_MODE_RENDERLESS	Graphics are not used or displayed and there is no window management.

Example

```
const char *CitrixServer      = "qaccitrix";
const int   CitrixOutputMode  = OUTPUT_MODE_NORMAL;

...

CtxConnect(CitrixServer, CitrixOutputMode);
```

CtxDisconnect

Disconnects from the Citrix server.

Syntax

void CtxDisconnect();

Return Value

None

Parameters

None

Example

```
CtxDisconnect();
```

CtxSetDomainLoginInfo

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

Syntax

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

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);
```

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);

}
```

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);
```

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, CtxModifierKeyEnum ModifierKeys, 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	CtxModifierKeyEnum 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, long button, long ModifierKeys, long Xpos, long Ypos);

Return Value

Parameters

Parameter	Description
windowInfo	Pointer to a Citrix Window Information object containing window data.
button	A mouse button to use for this action. NONE: No mouse button was specified L_BUTTON: The left mouse button R_BUTTON: The right mouse button M_BUTTON: The middle mouse button
ModifierKeys	A keyboard modifier to use for this action. NONE: No keyboard modifier was specified SHIFT: Shift key CONTROL: Control key ALT: Alt key EXTENDED: An extended key
Xpos	Move the mouse to this X coordinate.
Ypos	Move the mouse to this Y coordinate.

Example

```
CtxWI *CWI_2006c = new CtxWI(0x40034, "blah", 303, 208, 418, 145);
...
```

```
CtxMouseDown(CWI_2006c, L_BUTTON, NONE, 274, 316);
DO_MSLEEP(109);
CtxMouseUp(CWI_2006c, L_BUTTON, NONE, 274, 316);
```

CtxPing

Sends a ping request and wait for the response.

Syntax

void CtxPing(const char* identifier);

Return Value

Parameters

Parameter	Description
`identifier`	A string to send to the server.

Example

```
CtxPing("7");
```

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

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

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`	Event string.

Example

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

     if(CtxScreenEventExists(EVT_STR_CTXSCREENUPDATE,3000,"0 224 3910"))
     BeginBlock();
          RR__printf("Screen Update Found Test1");
     EndBlock();
```

CtxSetApplication

Connects to the Citrix server with the specified application name and application working directory name.

A session is created on the server where only the specified application appears. The working directory parameter is optional. Specify NULL for no working directory.

Syntax

```
void CtxSetApplication (const char* appName, const char* dirName);
```

Return Value

Parameters

Parameter	Description
`appName`	The application to start up after connecting.
`dirName`	The working directory for the application.

Examples

```
//Create a session containing only the application called
// application.exe. No working directory is specified.

CtxSetApplication("c:\\stuff\\application.exe", NULL);
```

CtxSetCitrixPort

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

Syntax

void CtxSetCitrixPort (int port);

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);
```

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 );
```

CtxSetICAFile

Uses the specified ICA file when connecting to a published application or desktop.

The ICA file can be used to specify a number of configuration options. A Citrix MetaFrame administrator can provide an ICA file for your environment.

Syntax

void CtxSetICAFile (const char *filename);

Return Value

Parameters

Parameter	Description
`filename`	The unqualified name or URL for the ICA file to use.

Example

```
CtxSetICAFile ("published-app.ica");
```

CtxSetLoginInfo

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

Syntax

void CtxSetLoginInfo (const char *username, const char *password);

Return Value

Parameters

Parameter	Description
`username`	The user name to use when connecting.
`password`	The password 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";
...
CtxSetLoginInfo (CitrixUsername, CitrixPassword);
```

CtxSetPingTimeout

Sets the number of seconds to wait for a ping to be acknowledged.

Syntax

void CtxSetPingTimeout (int timeout);

Return Value

Parameters

Parameter	Description
`timeout`	The number of seconds to wait for the ping to be acknowledged.

Example

```
CtxSetPingTimeout (30);
```

CtxSetWaitPointTimeout

Sets the number of seconds to wait for a wait point.

Syntax

void CtxSetWaitPointTimeout (int timeout);

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 WaitForWindowCreate method of the Citrix Window Information object.

This name is used for comparison if wildcards have been enabled by the SetEnableWildcardMatching method call earlier in the script.

Syntax

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

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.
```

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 activated and destroyed.

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);
```

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 `KeyDown(key)` and `KeyUp(Key)`.

Syntax

void CtxTypeChar(const CtxWI *wi, long vk, int mod);

Return Value

Parameters

Parameter	Description
`wi`	Pointer to a Citrix Window Information object containing window data.

vk	ASCII character to type.
mod	A keyboard modifier to use for this action.

Example

```
CtxTypeChar(CWI_3001e, 'F', ALT);   //Send ALT-F
```

CtxTypeVK

Sends the VK code that corresponds to a key typed by the user.

Inputs the key stroke specified by the key argument, which corresponds to the VK code. Ensure that `wi` is the foreground window. This is equivalent to `KeyDown(key)` and `KeyUp(Key)`.

Syntax

```
void CtxTypeVK(const CtxWI *wi, CtxKeyModifierEnum key, int 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	Amount of time to wait for the event.

Example

```
DO_MSLEEP (234);
// Wait for the title of the window that matches CWI_11 to change to "new title"
CtxWaitForCaptionChange (CWI_11, "new title", 2000);
DO_MSLEEP (125);
```

CtxWaitForScreenUpdate

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

Syntax

void CtxWaitForScreenUpdate(long x, long y, long w, long h, long nmWait);

Return Value

Parameters

Parameter	Description
x	X coordinate
y	Y coordinate
w	Width of the screen update
h	Height of the screen update
nmWait	Amount of time in milliseconds to wait for the event

Example

```
CtxWaitForScreenUpdate(154, 154, 253, 261, 500);
```

CtxWaitForWindowActivate

Waits for the specified window to be activated (brought to the foreground).

Syntax

void CtxWaitForWindowActivate(const CtxWI *wi, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

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

CtxWaitForWindowCreate

Waits for the specified window to be created.

Syntax

void CtxWaitForWindowCreate(const CtxWI *wi, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

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

CtxWaitForWindowDestroy

Waits for the specified window to be destroyed.

Syntax

void CtxWaitForWindowDestroy(const CtxWI *wi, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

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

CtxWaitForWindowLgIconChange

Waits for the specified window's 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	Amount of time in milliseconds to wait for the event.

Example

```
// Window CWI_13 ("Windows Task Manager") created 1101909450.125
CtxWaitForWindowCreate(CWI_13, 110);
CtxPoint(327, 2);  //1101909452.531
CtxWaitForWindowLgIconChange(CWI_13, "c48b65c8b825324a5ff73638118fb8fc", 1218);
```

CtxWaitForWindowMinimize

Waits for the specified window to be minimized.

Syntax

void CtxWaitForWindowMinimize(const CtxWI *wi, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
nmWait	Amount of time in milliseconds to wait for the event.

Example

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

CtxWaitForWindowMove

Waits for the specified window to be moved to the specified coordinates.

Syntax

void CtxWaitForWindowMove(const CtxWI *wi, long x, long y, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data
x	X coordinate
y	Y coordinate
nmWait	Amount of time in milliseconds to wait for the event

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowMove(CWI_40034, 132, 297, 2000);
```

CtxWaitForWindowResize

Waits for the specified window to be resized to the specified dimensions.

Syntax

void CtxWaitForWindowResize(const CtxWI *wi, long w, long h, long nmWait);

Return Value

Parameters

Parameter	Description
wi	Pointer to a Citrix Window Information object containing window data.
w	X coordinate.
h	Y coordinate.
nmWait	Amount of time in milliseconds to wait for the event.

Example

```
CtxWI *CWI_40034 = new CtxWI(0x40034, "Please wait...", 303, 208, 418, 145);
...
CtxWaitForWindowResize(CWI_40034, 100, 200, 500);
```

CtxWaitForWindowSmIconChange

Waits for the specified window's 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	Amount of time in milliseconds to wait for the event.

Example

```
// Window CWI_13 ("Windows Task Manager") created 1101909450.125
CtxWaitForWindowCreate(CWI_13, 110);
CtxPoint(327, 2);  //1101909452.531
CtxWaitForWindowSmIconChange(CWI_13, "924f75dd0db6ecb28d3e513053a8038e", 1391);
```

CtxWaitForWindowStyleChange

Waits for the specified window's style to be changed as specified.

Syntax

void CtxWaitForWindowStyleChange(const CtxWI *wi, long style, long extendedStyle, long nmWait);

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	Amount of time in milliseconds to wait for the event.

Example

```
Point (155, 1);
DO_MSLEEP (515);
//Wait for the window that matches CWI_11 to maximize
CtxWaitForWindowStyleChange (CWI_11, 0x14ca0044, 0x50100, 500);
DO_MSLEEP (11047);
```

CtxWindowEventExists

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

Syntax

BOOL CtxWindowEventExists(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/DB2 driver. It should only be called once at the end of a script.

DO_initODBC
Initializes QALoad's internal ODBC variables. Must be called at the beginning of the script prior to any other calls.

DO_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 are also present in DB2. They offer a way of tracking column metadata. Descriptors can be used for a number of different purposes, and can be shared by different statements. In most cases, an application doesn't require access to descriptors; however, in some cases accessing descriptors can simplify a number of operations.

There are four types of descriptors:

- Application Parameter Descriptor (APD)
  Contains either the input parameters set up by the application or the output columns following the execution of a CALL statement within SQL.

- Application Row Descriptor (ARD)
  Contains the row data as the row is presented to the application.

- Implementation Row Descriptor (IRD)
  Contains the row as it comes from the database.

- Implementation Parameter Descriptor (IPD)
  Contains the parameter elements after conversion heading to the database.

For more information on ODBC descriptors, refer to your ODBC 3.0 Programmer's Reference Volume and SDK Guide.

Handling connection descriptors

It is important to note that QALoad processes descriptor handles in the same way it processes statement handles. Each connection handle is associated with a unique descriptor handle. Each time a descriptor is allocated during conversion, QALoad associates descriptors with connections the same way that ODBC and DB2 do.

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

Parameter	Description

StatementIndex	Index into the table of ODBC statement handles.
ColumnNum	Number of the result set column to bind.
CDataType	ODBCSQLCDataTypeEnum 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_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
BufferLength	The length of the buffer for the data that is being returned.
PBufferLength	The length/indicator buffer to bind to the column.

Example

```
BEGIN_TRANSACTION();

DO_SQLAllocHandle( SQL_HANDLE_DBC, 0, C0 );
DO_SQLConnect( C0, "FHLOADDB2", "sa", "" );
DO_SQLAllocHandle( SQL_HANDLE_STMT, C0, S0 );
DO_SQLSetStmtAttr( S0, SQL_ATTR_ROW_ARRAY_SIZE, 5, SQL_IS_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 )
{
pReturnValue = DO_SQLGetData( S0, 1, SQL_C_LONG, 4 );
free(pReturnValue);
}
DO_SQLFreeHandle( SQL_HANDLE_STMT, S0 );
```

DO_SQLFreeConnect

Performs the cleanup of connection handles for ODBC 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

Parameter	Description
StatementIndex	Index into the table of statement handles.
SQLType	ODBCSQLGetTypeSQLTypeEnum 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
	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 Determines whether to return special columns that have NULL values. Valid 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 ) );
```

General Oracle

General Oracle Commands

DO_free_data
A required cleanup routine inserted into a script when it is generated and called before exiting the script. It should not be modified or moved.

DO_freeitem
Frees the memory associated with an ActiveData for Oracle variable. The ActiveData variable (source variable) may have been assigned by DO_GetSelectData, DO_OCI8GetSelectData, DO_strdup or DO_GetOutputData.

DO_GetOutputData
Copies the data from a bind variable into a source variable. Use with both Oracle 7 and 8 binds.

DO_makedate
Binds dates into a SQL statement in C-based scripts

DO_strdup
Places the data retrieved by ActiveData for Oracle from a QALoad central datapool or a local datapool.

DO_free_data

A required cleanup routine inserted into a script when it is generated and called before exiting the script. It should not be modified or moved.

Syntax

```
DO_free_data();
```

Return Value

Parameters

None.

DO_freeitem

Frees the memory associated with an ActiveData for Oracle variable.

The ActiveData variable (source variable) may have been assigned by DO_GetSelectData, DO_OCI8GetSelectData, DO_strdup or DO_GetOutputData.
The memory assigned for variables by ActiveData for Oracle is allocated from the program's free memory area (malloc). This function releases that memory. It is automatically placed in your script for all variables created by the Oracle conversion program.

If you add your own variables, you should include a DO_freeitem() to release allocated memory at the end of the transaction loop.

Syntax

```
DO_freeitem( char** name );
```

Return Value

Parameters

Parameter	Description
name	Pointer to source variable.

DO_GetOutputData

Copies the data from a bind variable into a source variable. Use with both Oracle 7 and 8 binds.

Source variables are created by ActiveData for Oracle so that postbind or fetch data from one portion of a script can be used as input to subsequent bind statements.

DO_GetOutputData() allocates memory for the contents of the source from the system's free memory pool (malloc).

If the formatType is INT_FORMAT, then the data is converted to an integer before formatting (using atol()). This implies that the formatString contains a %i, %d or equivalent.

Syntax

```
DO_GetOutputData( char** srcName, sword type, char* bindName, int length,
OracleFormatDataTypeEnum formatType, char* formatString, int addConstant );
```

Return Value

Parameters

Parameter	Description
srcName	Pointer to the address of a source variable. The function allocates memory for the source value, and copies its value into this variable. Note that this parameter is a char **.
type	External datatype of the bindName. Presently, only character and numeric data types are supported. Binary dates and rowID are not.
bindName	Pointer to a variable that contains the bind data to be copied.
length	Length of the bind data.
formatType	*OracleFormatDataTypeEnum* Data type to be used in the special format string. Acceptable values are: Value — Description _NONE_ — No special formatting. INT_FORMAT — Convert bind data to an integer before formatting. STRING_FORMAT — Assume that the bind data is not numeric.
formatString	A printf-style format. The data is formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the bindName before conversion.

Example

```
DO_OCIStmtExecute( HNDL(6) ); /* Exec for statement 6 */
DO_GetOutputData(&PB_XHOME_TELE, _FIXED_CHAR,
          CHAR_2_XHOME_TELE_9, *pAlen[65],
          _NONE_, "", 0);
DO_GetOutputData(&PB_NEXT_ID, _FIXED_CHAR,
          CHAR_2_ID, *pAlen[65], INT_FORMAT,
          "%04i", ADD(1) );
```

DO_makedate

Binds dates into a SQL statement in C-based scripts

Since standard date formats differ between countries, QALoad implements a language/ country independent method of binding dates into a SQL statement. When DO_makedate is used in conjunction with a DO_BindV, dates are properly processed regardless of the currently selected date format. If QALoad detects a bind variable representing a date, it automatically declares a variable of type ORADATE and generates the appropriate DO_makedate call.

Syntax

```
DO_makedate( ORADATE* date_var, int year, int month, int day, int hour, int min, int second
);
```

Return Value

Parameters

Parameter	Description
date_var	Pointer to a variable of type ORADATE.
year	Four-digit year.
month	Two-digit month.
day	Two-digit day of the month.
hour	Hour in the day (0 to 23).
min	Minute within the hour (0-59).
second	Seconds within the minute (0-59).

Example

This example shows how to get a date ready for use in a SQL statement:

```
ORADATE DATE_0_8;
...
DO_makedate( &DATE_0_8, 1995, 7, 12, 0, 0, 0 );
DO_BindV (CDA(0); "(text*):8, _DATE,7,NULL, (ub1*) &DATE_0_8, (ub1*) &DATE_0_8);
```

DO_strdup

Places the data retrieved by ActiveData for Oracle from a *QA*Load central datapool or a local datapool.

Versions

Versions of DO_strdup are:

DO_strdup(char** progVar, GET_DATA_FIELD(datapool_nbr, field_nbr));

DO_strdup(char** progVar, VARDATA(field_nbr));

Oracle OCI Version 7

Oracle OCI Version 7 Commands

DO_autocommitoff

Disables the automatic commit of every SQL data manipulation command.

DO_autocommiton

Enables the automatic commit of every SQL data manipulation command.

DO_binddate
Binds a date variable to a bind variable in a SQL statement.

DO_BindForUpdateRowID
Binds a rowID in an UPDATE or DELETE statement where the rowid originates from a previous SELECT FOR UPDATE statement.

DO_bindnull
Binds a NULL value to a bind variable in a SQL statement.

DO_bindstring
Binds a program variable to a bind variable in a SQL statement.

DO_BindV
Binds a program variable to a bind variable in a SQL statement.

DO_cleanup
Deallocates cursors, logon structures and allocated memory when the script is aborted.

DO_commit
Commits the current transaction.

DO_FetchIters
Identifies the number of fetch iterations that will be applied to the succeeding fetch loop. The Fetch Loop will execute (n) times according to the fetch iteration value.

DO_get_select_variable
Places the select-list item data recently fetched by DO_process_select_list in a program variable, which may then be used in subsequent statements.

DO_GetSelectData
Copies the data retrieved from a SQL SELECT statement into a program variable.

DO_init_alen
A routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_ScalarBindA.

DO_init_data
A routine that allocates and initializes all of the logon data areas, cursor data areas, structures, and so on which are used to run the script.

DO_init_indp
A routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_Bindv and DO_ScalarBindA calls.

DO_oclose
Disconnects a previously opened cursor, returning all resources back to the Oracle server.

DO_oexec
Executes the SQL statement associated with a cursor.

DO_olog
Establishes a connection between QALoad and an Oracle database.

DO_ologof
Closes a connection to the Oracle server, freeing its resources.

DO_oopen
Opens a cursor to the database.

DO_oopt
Sets rollback options for non-fatal errors on multi-row INSERT and UPDATE SQL statements and determines whether to wait for requested resources or return errors.

DO_oparse
Parses a SQL statement or a PL/SQL block and associates it with a cursor index.

DO_process_select_list
Fetches select-list data from the Oracle database. It is generally called repeatedly until there are no more rows satisfying the SQL select request.

DO_rollback
Rolls back the current transaction.

DO_ScalarBindA
Binds a program variable to a bind variable in a SQL statement.

DO_SoftClose
Closes a cursor without destroying its resources on the server.

DO_autocommitoff

Disables the automatic commit of every SQL data manipulation command.

Syntax

```
DO_autocommitoff(LDAIndex );
```

Return Value

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

```
ocof
```

Example

```
DO_autocommitoff( LDA(0) );
```

DO_autocommiton

Enables the automatic commit of every SQL data manipulation command.

Syntax

```
DO_autocommiton( LDAIndex );
```

Return Value

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

ocon

Example

```
DO_autocommiton( LDA(0) );
```

DO_binddate

Binds a date variable to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_binddate commands must be placed between the DO_oparse and the DO_oexec commands. To bind by position, instead of by name, preface the position with an @symbol.

Syntax

```
DO_binddate( cursor, name, &oradate_structure );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
name	Pointer to the name of the bind variable (null terminated string).
oradate_structure	An ORADATE structure. This structure is defined in the header files provided by Oracle. The '&' is the C address operator which specifies that the address or pointer to an ORADATE structure is being passed.

	year: Four-digit year. month: Two-digit month. day: Two-digit day of the month. hour: Hour in the day (0 to 23). min: Minute within the hour (0-59). second: Seconds within the minute (0-59).

Equivalent OCI

obndrv

Example

This example shows a Select command with one bind variable:

```
/* ORADATE declarations follow */

ORADATE DATE_0_6;
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE HIRE_DATE >:hiredate " );
DO_makedate( &DATE_0_6, 1980, 12, 17, 0, 0, 0 );
DO_binddate( CDA(0), ":hiredate", &DATE_0_6 );
DO_oexec( CDA(0) );
```

DO_BindForUpdateRowID

Binds a rowID in an UPDATE or DELETE statement where the rowID originates from a previous SELECT FOR UPDATE statement.

Syntax

```
DO_BindForUpdateRowID( cursor1, cursor0, bind_variable_name );
```

Return Value

Parameters

Parameter	Description
cursor1	Cursor index.
cursor0	Cursor index of the FOR UPDATE statement.
bind_variable	Name of a rowid bind variable.

Example

```
DO_oopen( LDA(0), CDA(0) );
DO_oparse( CDA(0), "SELECT EMPNO, ENAME FROM EMP FOR UPDATE OF EMPNO, ENAME");
DO_oexec( CDA(0) );

n = DO_process_select_list( CDA(0), 3 );

DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1), "UPDATE EMP SET EMPNO=:empno, ENAME=:ename WHERE ROWID = :row_id");
DO_BindForUpdateRowID( CDA(1), CDA(0), ":row_id" );
```

```
DO_BindV(CDA(1), (text*)":empno",_VARCHAR2,
           4, NULL, (ub1 *) "7421",
           (ub1 *) VARCHAR2_0_empno_0);
DO_BindV(CDA(1), (text*)":ename",_VARCHAR2,
           4, NULL, (ub1 *) "WARD",
           (ub1 *) VARCHAR2_0_ename_1);
DO_oexec( CDA(1) );
```

DO_bindnull

Binds a NULL value to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). The input value is set null. Bind statements must be placed after the DO_oparse and before the DO_oexec. DO_bindnull cannot be used with bind variables that return results, as in stored procedures OUTPUT parameters.

Syntax

```
DO_bindnull( cursor, name );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
name	The name of the bind variable as a null-terminated character string.

Equivalent OCI

```
obndrv, obndrn
```

Example

This example shows a Select command with two bind variables, :empid and :id, which are being bound to a NULL value.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID =:empid AND EMP_DEPT_ID =
:id " );
DO_bindnull( CDA(0), ":empid" );
DO_bindnull( CDA(0), ":id" );
DO_oexec( CDA(0) );
```

DO_bindstring

Binds a program variable to a bind variable in a SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:).

DO_bindstring must be called after the DO_oparse and before the DO_oexec. Once you have bound a variable, you can change the value and length and then call another DO_oexec.

DO_bindstring only supports the binding of strings, nulls, and dates. If you need to bind a numeric value, convert it first to a string before passing it to DO_bindstring. If needed, Oracle automatically converts character data types to numeric.

Note: DO_bindstring is a deprecated command. Use DO_BindV or DO_ScalarBindA. DO_bindstring binds every data type as a fixed character and forces the Oracle server to make implicit database conversions. Also, you must variablize OUTPUT variables or they overwrite the input data held by string constants.

Syntax

```
DO_bindstring( cursor, name, value );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
name	Pointer to the name of the bind variable (null terminated).
value	Pointer to a string containing the value for the bind variable (null terminated).

Equivalent OCI

```
obndrv, obndrn
```

Example

This example shows a Select command with two bind variables, :empid and :id.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID =:empid AND EMP_DEPT_ID =
:id " );
DO_bindstring( CDA(0), ":empid", "200" );
DO_bindstring( CDA(0), ":id", "100" );
DO_oexec( CDA(0) );
```

DO_BindV

Binds a program variable to a bind variable in a SQL statement.

A DO_BindV is generated wherever an obndrv occurred in the capture file. DO_BindV accurately reproduces the original bind call made by the application. This eliminates extra data conversion steps and improves handling of OUTPUT variables to Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:). DO_BindV must be called after the DO_oparse and before a DO_oexec. Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable.

Currently, DO_BindV is not supported for cursor, mlslabel, packed-decimal, oslabel, PCC-descriptor, and the new Oracle 8 datatypes.

Syntax

```
DO_BindV(index, name, type, progvl, indp, input, progv);
```

Return Value

Parameters

Parameter	Description
index	Cursor table index.
name	Pointer to name of the bind variable (null terminated).
type	External datatype of bind variable.
progvl	Size of progv. This is the maximum size of the buffer. If binding an OUTPUT variable, progvl must be at least as large as the expected output value.
indp	Pointer to a null indicator variable: plndp[0] points to make_indp[0], and make_indp holds the value of the null indicator. If make_indp[0]=SET_NULL, the input will be passed to Oracle as null. Otherwise, data is passed as shown in the bind call.
input	Pointer to buffer containing input data.
progv	Output data buffer.

Equivalent OCI

```
obndrv, obndrn
```

Example

This example shows a Select command with two bind variables, :empid and :id.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID = :empid AND EMP_DEPT_ID =
:id" );

make_indp[0]=0

DO_BindV(CDA(0), ":empid", _STRING, 48, pIndp[0], "200", STRING_0_empid_3);

make_indp[1]=0

DO_BindV (CDA(0), ":id", _STRING, 48, pIndp[1], "100", STRING_0_id_3);
DO_oexec ( CDA(0) );
```

DO_cleanup

Deallocates cursors, logon structures, and allocated memory when the script is aborted.

Note: Do not modify or move this command.

Syntax

```
sword DO_cleanup();
```

Return Value

Parameters

None.

DO_commit

Commits the current transaction.

Syntax

```
DO_commit( LDAIndex );
```

Return Value

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

```
ocom
```

Example

```
DO_commit( LDA(0) );
```

DO_FetchIters

Identifies the number of fetch iterations that are applied to the succeeding fetch loop. The Fetch Loop executes (*n*) times according to

the fetch iteration value.

The fetch iteration value is derived from the script capture's fetch iteration data for each Select statement. However, the Fetch Iteration Override in Oracle Convert Options may be used to replace all fetch iteration values in the script. The override range is 1-1000000. The default value for each convert activity is 0 (no override).

Syntax

```
DO_FetchIters( cursorIndex, fetchIterationValue );
```

or

```
DO_FetchIters( statementHandleIndex, fetchIterationValue );
```

Return Value

Parameters

Parameter	Description
cursorIndex	An index to an allocated OC17 cursor or Oracle 8 statement handle used in the previous call to oparse, osql3, upipse, upiosq, or OCIStmtPrepare.
fetchIterationValue	Number of iterations to be applied to the succeeding fetch-loop.
statementHandleIndex	An index to an allocated OC17 cursor or Oracle 8 statement handle used in the previous call to oparse, osql3, upipse, upiosq, or OCIStmtPrepare.

Example

The following OC17 example shows how DO_FetchIters is used relative to the parse and fetch-loop:

```
DO_oparse( CDA(0),"select ename, empno, mgr from emp" );
DO_FetchIters( CDA(0), ITERS(4) );
DO_oexec( CDA(0) );

while (DO_process_select_list( CDA(0), 1 ))
//1 = the number of rows per iteration
{
}
```

The following OCI18 example shows how DO_FetchIters is used relative to the prepare and fetch-loop.

```
DO_OCIStmtPrepare( HNDL(6), "select empno from emp", OCI_NTV_SYNTAX );
DO_FetchIters( HNDL(6), ITERS(13) );
DO_OCIDefine( HNDL(6), HNDL(1), 1, 1,_VARCHAR2, 4, IS_ATTRIBUTE );
DO_OCIStmtExecute( HNDL(8), HNDL(6), HNDL(1), 1, OCI_DEFAULT );

while (DO_OCIProcessSelectList( HNDL(6), 1) )
//1 = the number of rows per iteration
{
DO_OCI8GetSelectData(FETCH(1), COL(1), ROW(1), &FD_stmnt_1_col_1_row_1, _NONE_, "", 0 );
}
```

DO_GetSelectData

Copies the data retrieved from a SQL SELECT statement into a program variable.

DO_GetSelectData processes the data retrieved by DO_process_select_list() by copying the value of the fetched data to another program variable. Typically, the program variable is also a source variable. Source variables are created by ActiveData for Oracle so that postbind and/or fetch data from one portion of a script can be used as input to subsequent bind statements.

Note: If you are working with Oracle 8 select output data, use DO_OCI8GetSelectData instead.

If the formatType is INT_FORMAT, then the data is converted to an integer before formatting (using atol()). This implies that the formatString contains a %i, %d or equivalent.

Syntax

```
DO_GetSelectData( fetchCount, colnum, rowNum, srcName, formatType, formatString,
addConstant);
```

Return Value

Parameters

Parameter	Description
fetchCount	A number from 1-n indicating which fetch sequence to use to fetch the data. The script code for a fetch statement is generally output as a C-based while-loop. This loop will retrieve data until no more data is available. This parameter determines which iteration of that loop to use to retrieve the data.
colnum	Column number to use to fetch the data. The first column is 1.
rowNum	Row number to use to fetch the data. The first row number is 1.
srcName	Pointer to the address of a source variable. The function will allocate memory for the source value and copy its value into this variable. Note that this parameter is a char **.
formatType	Data type to be used in the special format string. Acceptable values are: _NONE_, No special formatting. INT_FORMAT, Convert bind data to an integer before formatting. STRING_FORMAT, Assume that the bind data is numeric.
formatString	A printf-style format. The data will be formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the fetch data before conversion.

Example

The following example copies the fetched value of the first select-list item of the second row (in the first fetch iteration which retrieves 409 rows) to program variable FD_stmnt_3_col_1_row_2.

It also copies the fetched value of the fifth select-list item of the second row (in the first fetch iteration) to program variable FD_stmnt_3_col_5_row_2.

```
DO_oexec( CDA(1) );  /* Exec for statement 3 */

while ( DO_process_select_list( CDA(1), 409 ) )
{
DO_GetSelectData( FETCH(1), COL(1), ROW(2), &FD_stmnt_3_col_1_row_2, _NONE_, "", 0 );
DO_GetSelectData( FETCH(1), COL(5), ROW(2), &FD_stmnt_3_col_5_row_2,_NONE_, "", 0 );
} /* end of DO_process_select_list */
```

DO_get_select_variable

Places the select list item data recently fetched by DO_process_select_list in a program variable, which may then be used in subsequent statements.

Note: DO_process_select_list is called repeatedly in a loop. Each call to DO_process_select_list fetches a number of rows into the script's internal buffer. The number of rows is specified in the second parameter of DO_process_select_list. DO_get_select_variable, in turn, copies the fetched data from a specific row and the select list item into a program variable.

All select list items are converted by the server into a null terminated string format prior to being processed by your script. Therefore, dates and numbers appear as readable ASCII character strings.

The program does not check to verify that the length of the buffer is sufficiently large to contain the returned value.

Syntax

```
DO_get_select_variable( pos, row, value );
```

Return Value

Parameters

Parameter	Description
pos	Variable to retrieve (starts at 1).
row	Row in the buffer to retrieve (starts at 1).
value	Pointer to a character array into which the data is placed.

Example

This example shows how the first select-list item from the second fetched row is copied to the program variable coname.

```
char coname[128];
:
:
DO_oexec( CDA(0) ); /* Exec for statement 2 */
while ( DO_process_select_list( CDA(0), 30 ) )
{
DO_get_select_variable( 1, 2, coname );
}
```

DO_init_alen

A routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_ScalarBindA.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function is not always called; for example, a script may not contain any DO_ScalarBindA calls, or the bind calls that are contained in the script do not utilize pAlen. This function should not be moved or modified.

Syntax

```
DO_init_alen(make_alen,pAlen,ALEN_COUNT);
```

Return Value

Parameters

Parameter	Description
make_alen	A pointer to an array that holds the values of the length of data.
pAlen	A pointer to make_alen. Each element holds the pointer to the corresponding make_alen. In the example pAlen[0] = &make_alen[0], the contents of make_alen[0]are assigned before the call to DO_ScalarBindA.
ALEN_COUNT	The number of pAlen utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define ALEN_COUNT at the beginning of the script. Every bind does not necessarily utilize a pAlen. For example, if the alen was captured as NULL, NULL replaces the use of pAlen.

Example

```
#define ALEN_COUNT 20

:
sb2* pIndp[ALEN_COUNT]; /* sb2 is a signed integer */
sb2 make_alen[ALEN_COUNT];
:
DO_init_alen( make_alen, pAlen, ALEN_COUNT );
:
make_indp[1]=0;
make_alen[0]=90;
DO_ScalarBindA( CDA(3), ":id", _STRING, -1, pAlen[0], pIndp[1],"id", STRING_3_id_69 );
```

DO_init_data

A routine that allocates and initializes all of the logon data areas, cursor data areas, structures, and so on, which are used to run the script.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. It should not be modified or removed.

Syntax

```
DO_init_data(s_info, LOGON_COUNT, CURSOR_COUNT,HANDLE_COUNT, DESCRIPTOR_COUNT);
```

Return Value

Parameters

Parameter	Description
s_info	Structure used by each virtual user.
LOGON_COUNT	The number of logons in the script. If this number is incorrect, the script will fail. The number can be modified in the #define LOGON_COUNT at the beginning of the script.
CURSOR_COUNT	The number of cursors opened in the script. If this number is

	incorrect, the script will fail. The number can be modified in the #define CURSOR_COUNT at the beginning of the script.
HANDLE_COUNT	The number of handles opened in the script. If this number is incorrect, the script will fail. The number can be modified in the #define HANDLE_COUNT at the beginning of the script.
DESCRIPTOR_COUNT	The number of descriptors opened in the script. If this number is incorrect, the script will fail. The number can be modified in the #define DESCRIPTOR_COUNT at the beginning of the script.

Example

```
#define LOGON_COUNT 5
#define CURSOR_COUNT 35
#define HANDLE_COUNT 9
#define DESCRIPTOR_COUNT 1
:
:
DO_init_data( s_info, LOGON_COUNT, CURSOR_COUNT, HANDLE_COUNT, DESCRIPTOR_COUNT);
```

DO_init_indp

A routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_Bindv and DO_ScalarBindA calls.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function should not be moved or modified.

Syntax

```
DO_init_indp(make_indp,pIndp,INDP_COUNT);
```

Return Value

Parameters

Parameter	Description
make_indp	A pointer to an integer array that holds the values of the null indicator variable.
pIndp	A pointer to make_indp. Each element holds the pointer to the corresponding null indicator variable. In the example pIndp[0] = &make_indp[0], the contents of make_indp[0] is assigned before the call to DO_BindV or DO_ScalarBindA.
INDP_COUNT	The number of indicator variables utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define INDP_COUNT at the beginning of the script. Every bind does not necessarily utilize a pIndp. If the null indicator was captured as NULL, NULL replaces the use of pIndp.

Example

```
#define INDP_COUNT 20
sb2* pIndp[INDP_COUNT]; /* sb2 is a signed integer */
sb2 make_indp[INDP_COUNT];
DO_init_indp(make_indp,pIndp,INDP_COUNT);
```

DO_oclose

Disconnects a previously opened cursor, returning all resources back to the Oracle server.

Syntax

```
DO_oclose( cursor );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.

Equivalent OCI

```
oclose
```

Example

```
DO_oclose( CDA(0) );
```

DO_oexec

Executes the SQL statement associated with a cursor.

Before calling DO_oexec, the SQL statement must be parsed by calling DO_Oparse using the same cursor.

Syntax

```
DO_oexec( cursor );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.

Equivalent OCI

```
oexec
```

Example

This example shows parsing and execution of a SQL statement:

```
DO_oparse( CDA(1),"select id, coname from company" );
DO_oexec( CDA(1) );
```

DO_olog

Establishes a connection between *QA*Load and an Oracle database.

An application must log in to Oracle before it can perform any other operations. Multiple connections to one or more Oracle instances is supported.

A user ID string is made up of the user's login ID, password, and an Oracle connection string.

A forward slash (/) separates the password from the user ID, and the connection string is preceded by the @symbol.

Syntax

```
DO_olog( LDAIndex, connect-string );
```

Return Value

Parameters

Parameter	Description
LDAIndex	Logon data area index.
connect-string	A null terminated string containing the Oracle login ID, password, and connection.

Equivalent OCI

```
orlon
```

Example

This example shows a typical Oracle login sequence:

```
DO_olog( LDA(0), "scott/tiger@domain" );
```

DO_ologof

Closes a connection to the Oracle server, freeing its resources.

Syntax

```
DO_ologof( LDAIndex );
```

Return Value

Parameters

Parameter	Description
LDAIndex	Logon data area index.

Equivalent OCI

ologof

Example

```
DO_ologof( LDA(0) );
```

DO_oopen

Opens a cursor to the database.

Processing commands such as DO_oparse and DO_oexec require an open cursor. There may be multiple open cursors at one time, so operations may be repeated without re-parsing the SQL statement. QALoad automatically manages cursor opens and closes.

Syntax

```
DO_oopen( IdaIndex, cursor );
```

Return Value

Parameters

Parameter	Description
IdaIndex	Logon data area index.
cursor	Cursor table index (first index is 0).

Equivalent OCI

oopen

Example

This example shows how a cursor is opened, a command is executed, and the cursor is subsequently closed:

```
DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1),"select id, coname from company" );
DO_oexec( CDA(1) );
DO_process_select_list( CDA(1), 100 ); /* get 100 rows */
DO_oclose( CDA(1) );
```

DO_oopt

Sets rollback options for non-fatal errors on multi-row INSERT and UPDATE SQL statements and determines whether to wait for requested resources or return errors.

Syntax

```
DO_oopt( cursor, rbopt, waitopt );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
rbopt	0 = Rollback on any error. 2 = Rollback only the failing row.
waitopt	0 = Wait indefinitely for resources to be available. 4 = Return an error if a resource is requested, but not available.

Equivalent OCI

```
oopt
```

DO_oparse

Parses a SQL statement or a PL/SQL block and associates it with a cursor index.

QALoad scripts use deferred mode linking and DO_oparse defers the parse. In this mode, SQL statements are not actually sent to the server until the DO_oexec call. Therefore, SQL syntax errors are not reported at DO_oparse, but rather at DO_oexec.

Syntax

```
DO_oparse( cursor, statement );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
statement	Pointer to null terminated string containing the SQL statement.

Equivalent OCI

```
oparse
```

Example

This example shows a complete parse, execute, and fetch cycle for a SQL Select statement:

```
DO_oopen( LDA(0), CDA(1) );
DO_oparse( CDA(1),"select id, coname from company" );
DO_oexec( CDA(1) );
DO_process_select_list( CDA(1), 100 ); /* get 100 rows */
DO_oclose( CDA(1) );
```

DO_process_select_list

Fetches select-list data from the Oracle database. It is generally called repeatedly until there are no more rows satisfying the SQL select request.

The first time DO_process_select_list retrieves data for a SQL statement, it loops through all the returned fields (using odescr) and builds up a set of internal buffers to store the returned data. All data is returned as ASCII strings.

Syntax

```
DO_process_select_list( cursor, rowcount );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor table index.
rowcount	Number of rows to fetch into the buffer.

Equivalent OCI

```
ofen, odescr, and odefin
```

Example

This example shows the DO_process_select_list being called repeatedly, so all the rows are read:

```
DO_oexec( CDA(0) ); /* Exec for statement 2 */
while ( DO_process_select_list( CDA(0), 30 ) ); /* Read all rows, 30 at a time. */
```

DO_rollback

Rolls back the current transaction.

Syntax

```
DO_rollback( ldaIndex );
```

Return Value

Parameters

Parameter	Description
ldaIndex	Logon data area index.

Equivalent OCI

orol

Example

```
DO_rollback( LDA(0) );
```

DO_ScalarBindA

Binds a program variable to a bind variable in a SQL statement.

A DO_ScalarBindA is generated wherever an obndra occurred in the capture file and the type of bind was a single (scalar) value, not an array. DO_ScalarBindA accurately reproduces the original bind call made by the application.

This eliminates extra data conversion steps and improves handling of OUTPUT variables to Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:).

DO_ScalarBindA must be called after the DO_oparse and before a DO_oexec. Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable.

Currently, DO_ScalarBindA does not support packed decimal, PCC-descriptor, cursor, mslable, oslabel, and any new Oracle 8 datatypes.

Syntax

```
DO_ScalarBindA (index, name, type, progvl, alen, indp, input, progv);
```

Return Value

Parameters

Parameter	Description
index	Cursor table index.
name	Pointer to name of the bind variable (null terminated).
type	External datatype of bind variable.
progvl	Size of progv. This is the maximum size of the buffer. If binding an OUTPUT variable, progvl must be at least as large as the expected output value.
alen	Pointer to variable representing length of data. palen[0] points

	to the value of make_alen[0].
indp	Pointer to a null indicator variable. plndp[0] points to the value of make_indp[0]; If make_indp=SET_NULL, null will be passed as the data for the input. Otherwise, the data is passed as shown in the bind call.
input	Pointer to buffer containing input data.
progv	Output data buffer.

Equivalent OCI

obndra

Example

This example shows a Select command with two bind variables, :empid and :id.

```
DO_oparse( CDA(0), "SELECT EMP_ID FROM EMP_TUTORIAL WHERE EMP_ID = :empid AND EMP_DEPT_ID =
:id" );
make_indp[0]=0
DO_ScalarBindA(CDA(3), ":empid", _STRING, -1, NULL, pIndp[0], "200", STRING_3_empid_69);
make_indp[1]=0
make_alen[0]=90;
DO_ScalarBindA(CDA(3), ":id", _STRING, -1, pAlen[0], pIndp[1],"id",STRING_3_id_69);
DO_oexec( CDA(0) );
```

DO_SoftClose

Closes a cursor without destroying its resources on the server.

If the application is in the deferred mode, the cursor is not actually closed but is placed on a cursor-free list on the client. During any subsequent open cursor calls, the free list is checked first to satisfy the request. A soft close reduces communication with the server because it does not cancel the cursor. A SQL statement associated with the cursor remains valid until the cursor is reused to pause another SQL statement.

Syntax

```
DO_SoftClose( cursor );
```

Return Value

Parameters

Parameter	Description
cursor	Cursor date area index.

Example

This example shows how to use a soft close to execute a SQL statement again, then to parse and execute another SQL statement.

```
DO_oopen( LDA(0), CDA(0) );
DO_oparse( CDA(0), "select * from emp");
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 ) );
```

```
/* Soft close cursor */
DO_SoftClose( CDA(0) );

/* Reuse cursor to repeat statement */
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 ) );

/* Reuse cursor to parse next statement */
DO_oparse( CDA(0), "select ename from emp where empno = 7788";);
DO_oexec( CDA(0) );
while( DO_process_select_list( CDA(0), 15 );
DO_oclose( CDA(0) );
```

Oracle OCI Version 8

Oracle OCI Version 8 Commands

DO_OCI8BindDate
Binds a date variable (created by DO_makedate) to a bind variable in a SQL statement.

DO_OCI8BindNull
Binds a NULL value to a bind variable in a SQL statement.

DO_OCI8BindString
Binds a program variable to a bind variable in a SQL statement.

DO_OCI8GetSelectData
Copies the data retrieved from an Oracle8 SQL SELECT statement into a program variable.

DO_OCI8InitAlen
An Oracle8-specific routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_OCIBind.

DO_OCI8InitIndp
An OCI8-specific routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_OCIBind.

DO_OCIAttrSet
Sets a particular attribute for a previously allocated Oracle 8 OCI handle.

DO_OCIBind
Binds a program variable to a bind variable in a SQL statement.

DO_OCICommit
Commits the current Oracle8 transaction. A commit should be performed after all relevant SQL statements have been processed.

DO_OCIDefine
Associates an item in a select-list to an Oracle external datatype and an output data buffer.

DO_OCIDescriptorAlloc
Allocates and initializes an Oracle 8 OCI descriptor or LOB locator.

DO_OCIDescriptorFree
De-allocates an Oracle 8 OCI descriptor or LOB locator.

DO_OCIEnvFreeAll
De-allocates all environment handles before the end of an OCI8 script.

DO_OCIEnvInit
Allocates and initializes an Oracle OCI8 environment handle.

DO_OCIExecute
Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCIStmtPrepare. Note that SQL syntax errors are reported at execution time.

DO_OCIHandleAlloc
Allocates and initializes an Oracle 8 OCI handle.

DO_OCIHandleFree
De-allocates an Oracle 8 OCI handle.

DO_OCIInitialize
Initializes the Oracle OCI8 process environment. This command must be issued once in a QALoad script prior to any other Oracle8 script commands, and should be outside any QALoad transactions.

DO_OCILdaToSvcCtx
Toggles an Oracle 7 logon data area to an Oracle 8 service context handle. This should be done after using DO_OCISvcCtxToLda to create Oracle 7 in a database session in Oracle 8.

DO_OCILobRead
Reads a LOB into a buffer.

DO_OCILobWrite
Writes the contents of a buffer into an Oracle 8 LOB.

DO_OCILogoff
Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

DO_OCILogoffEx
Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

DO_OCILogon
Creates a simple Oracle OCI8 logon connection and session for QALoad . Any application must log on to Oracle before performing any other Oracle operations.

DO_OCIProcessSelectList
Fetches select-list data from an Oracle 8 database after an OCIStmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

DO_OCIProcessSelectList_EX
Fetches select-list data from an Oracle 8 database after an OCIStmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

DO_OCIRollback
Rolls back the current Oracle8 transaction.

DO_OCIServerAttach
Creates a standard Oracle OCI8 database connection for QALoad . Note that individual Oracle 8 user logons are done with the DO_OCIServerAttach command.

DO_OCIServerDetach
Detaches QALoad from the Oracle OCI8 data source connection previously attached to with the DO_OCIServerAttach command. Note that all users must be logged off with the DO_OCISessionEnd command before this call.

DO_OCISessionBegin
Creates an Oracle OCI8 logon session for QALoad to a server previously attached to with DO_OCIServerAttach. Any application must log on to Oracle before performing any other Oracle operations.

DO_OCISessionEnd
Terminates an Oracle user session previously created with the DO_OCISessionBegin command.

DO_OCIStmtExecute
Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCIStmtPrepare. Note that SQL syntax errors are reported at execution time.

DO_OCIStmtPrepare
Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

DO_OCIStmtPrepare_EX
Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

DO_OCISvcCtxToLda
Toggles an Oracle 8 service context handle to an Oracle 7 logon data area. This allows Oracle 7 cursors to be created in a database session created in Oracle 8.

DO_OCITransCommit
Commits the current Oracle 8 transaction. A commit should be performed after all relevant SQL statements have been processed.

DO_OCITransRollback
Rolls back the current Oracle 8 transaction.

Logging on and off Oracle Net 8

Command sequence for logging in to Oracle 8

```
/* NOTE: HNDL(0) is the environment handle. It should be previously specified in a
DO_OCIEnvInit call */

/* An error handle is used for Oracle8 error handling. HNDL(1) is the index to the new
error(OCI_HTYPE_ERROR) handle. */
DO_OCIHandleAlloc( HNDL(0), HNDL(1), OCI_HTYPE_ERROR);

/* A server handle is allocated for DO_OCIServerAttach. HNDL(2) is the index to the new
server (OCI_HTYPE_SERVER) handle. */
DO_OCIHandleAlloc( HNDL(0), HNDL(2), OCI_HTYPE_SERVER);

/* The DO_OCIServerAttach handle uses the server (HNDL(2)) handle previously allocated in
DO_OCIHandleAlloc. Note that the TNS data source name is the third parameter in this call.
*/
DO_OCIServerAttach( HNDL(2), HNDL(1), "oracledb.world", 15, OCI_DEFAULT);

/* A service context handle is now allocated. HNDL(3) is the index to the new service
context (OCI_HTYPE_SVCCTX) handle. */
DO_OCIHandleAlloc( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);
```

```
/* The allocated service context handle (HNDL(3)) is now set as an attribute of the server
handle(HNDL(2)) in this DO_OCIAttrSet call. Note that the error handle (HNDL(1)) is a
parameter in a DO_OCIAttrSet call. */
DO_OCIAttrSet( HNDL(3), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(1), HNDL(2));

/* A session handle is allocated for DO_OCISessionBegin. HNDL(3) is the index to the new
session (OCI_HTYPE_SESSION) handle. */
DO_OCIHandleAlloc( HNDL(0), HNDL(4), OCI_HTYPE_SESSION);

/* The username is set as an attribute of the session handle (HNDL(4)). */
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),
IS_ATTRIBUTE);

/* The password is set as an attribute of the session handle (HNDL(4)). */
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),
IS_ATTRIBUTE);

/* The DO_OCISessionBegin call uses the service context handle (HNDL(3) and the session
handle (HNDL(4)). . Note that the error handle (HNDL(1)) is a parameter in a
DO_OCISessionBegin call. The Credentials parameter is OCI_CRED_RDBMS, which means that
username and password must have been explicitly set in previous calls to DO_OCIAttrSet. If
the user verification is integrated with external credentials, use OCI_CRED_EXT as this
value. When you use OCI_CRED_EXT, you wil not have to set the username and password in
DO_OCIAttrSet calls prior to DO_OCISessionBegin. */
DO_OCISessionBegin( HNDL(3), HNDL(1), HNDL(4), OCI_CRED_RDBMS, OCI_DEFAULT);
```

Command sequence for logging off Oracle 8

```
/* The DO_OCISessionEnd call uses the same service context handle (HNDL(3)) and session
handle (HNDL(4)) used in DO_OCISessionBegin. Note that the error handle (HNDL(1)) is a
parameter in a DO_OCISessionEnd call. */
DO_OCISessionEnd( HNDL(3), HNDL(1), HNDL(4), OCI_DEFAULT);

/* The session handle (HNDL(4)) is no longer needed, so it is de-allocated with the
DO_OCIHandleFree call. */
DO_OCIHandleFree( HNDL(4), OCI_HTYPE_SESSION);

/* The DO_OCIServerDetach call uses the same server handle (HNDL(2)) and service context
handle (HNDL(1)) used in the DO_OCIServerAttach call. Note that the error handle (HNDL(1))
is a parameter in a DO_OCIServerDetach call. */
DO_OCIServerDetach( HNDL(2), HNDL(1), OCI_DEFAULT);

/* The server handle (HNDL(2)) is no longer needed, so it is de-allocated with the
DO_OCIHandleFree call. */
DO_OCIHandleFree( HNDL(2), OCI_HTYPE_SERVER);

/* The service context handle (HNDL(3)) is no longer needed, so it is de-allocated with the
DO_OCIHandleFree call. */
DO_OCIHandleFree( HNDL(3), OCI_HTYPE_SVCCTX);

/* If the error handle (HNDL(1)) is no longer needed, it should be de-allocated with the
DO_OCIHandleFree call. */
DO_OCIHandleFree( HNDL(1), OCI_HTYPE_ERROR);
```

Using QALoad script commands to log on and off an Oracle8 database

1. Create the appropriate handles and increment the `HANDLE_COUNT` parameter in the *QA*Load script.

   a. Find the number after `HANDLE_COUNT` in the *QA*Load script. Note the current number, and then add four (4) to this number. Four is the count of the number of new handles we will be allocating for use by this logon and logoff example.
   For example if the following line is in the script:
   `HANDLE_COUNT 35`
   edit it to read:
   `HANDLE_COUNT 39`

b. Then, allocate a server handle, a service context handle, a session handle and an error handle (or you may use another pre-allocated error handle; this error handle must not be freed before all logoff commands are called).

c. Associate the numbers 35, 36, 37, and 38 (starting with the previous HANDLE_COUNT and adding 1 for each new handle) with these new handles to be allocated, as shown in the following example:

```
DO_OCIHandleAlloc( HNDL(36), OCI_HTYPE_SERVER);
DO_OCIHandleAlloc( HNDL(37), OCI_HTYPE_SVCCTX);
DO_OCIHandleAlloc( HNDL(38), OCI_HTYPE_SESSION);
DO_OCIHandleAlloc( HNDL(35), OCI_HTYPE_ERROR);
```

2. Add the call to attach to the Oracle database (DO_OCIServerAttach). The following examples shows the code for the DO_OCIServerAttach call that uses the handles allocated in the previous step:

```
DO_OCIServerAttach( HNDL(36), HNDL(35),"oracledb.world", 15, OCI_DEFAULT);
```

3. Set the allocated service context handle as an attribute to the server handle with calls to DO_OCIAttrSet. Setting the service context handle as an attribute of the server handle allows the service context handle to be used in the DO_OCISessionBegin call. Ensure the handle indexes are correct, and keep the other parameters the same as shown below:

```
DO_OCIAttrSet( HNDL(37), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(35), HNDL(36));
```

4. If you are using Oracle security, set the session handle attributes. You will need to specify the username and password. Using the DO_OCIAttrSet calls will tie the username and password as attributes to the session handle. Ensure that the UserName and UserNameLength parameters are set correctly for this first DO_OCIAttrSet call, and the Password and PasswordLength attributes are set correctly for the second DO_OCIAttrSet call, as shown below:

```
DO_OCIAttrSet( HNDL(38), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(35),
IS_ATTRIBUTE);
DO_OCIAttrSet( HNDL(38), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(35),
IS_ATTRIBUTE);
```

5. Start the Oracle session with a call to DO_OCISessionBegin, as shown below. Note that the OCI_CRED_RDBMS parameter implies that the username and password are set with DO_OCIAttrSet calls, if using integrated security, step 4 is not needed, and use OCI_CRED_EXT as the parameter:

```
DO_OCISessionBegin( HNDL(37), HNDL(35), HNDL(38), OCI_CRED_RDBMS, OCI_DEFAULT);
```

6. After all SQL statements for the session have completed, log off the database as follows:

a. End the session with the DO_OCISessionEnd call:

```
DO_OCISessionEnd( HNDL(37), HNDL(35), HNDL(38), OCI_DEFAULT);
```

b. Disconnect from the server with the DO_OCIServerDetach call:

```
DO_OCIServerDetach( HNDL(36), HNDL(35), OCI_DEFAULT);
```

c. Free the allocated handles using DO_OCIHandleFree, or a memory leak will develop in the application:

```
DO_OCIHandleFree( HNDL(38), OCI_HTYPE_SESSION);
DO_OCIHandleFree( HNDL(36), OCI_HTYPE_SERVER);
DO_OCIHandleFree( HNDL(37), OCI_HTYPE_SVCCTX);
DO_OCIHandleFree( HNDL(35), OCI_HTYPE_ERROR);
```

Oracle SQL statements in Oracle 8 with QALoad script commands

SQL Statement Types

QALoad scripts support the following standard SQL statements:

- SQL data manipulation language (DML) statements
  Examples of DML statements include:

```
ළ2-8 QSELECT * FROM USER_TAB;
INSERT INTO EMP (VALUES "John Doe", "Accounting", 20, 500.00);
DELETE FROM DEPT WHERE DEPTNO = 100;
UPDATE EMP SET DEPTNO = 40 WHERE DEPTNO = 50;
```

- Anonymous PL/SQL blocks
  An example of anonymous PL/SQL blocks includes:
  `BEGIN UPDATE EMP SET PAY = 400.00;  END;`

- SQL data definition language (DDL) statements
  An example of DDL statements includes:
  `CREATE TABLE emp (empno NUMBER(5) PRIMARY KEY);`

- PL/SQL stored procedure or function calls
  An example of a PL/SQL stored procedure call includes:
  `"BEGIN qaload_regtest.emptest(:pkey, :f1, " ":num2, :opkey, :of1, :onum2); END;`

Note that QALoad does not support Oracle 8 objects, Oracle 8 user-defined types (UDTs) or Oracle 8 reference pointers.

QALoad Script Command Sequence, Example 1: INSERT Statement

QALoad Script Command Sequence, Example 2: SELECT statement

QALoad Script Command Sequence, Example 3: PL/SQL Stored Procedure Call

Command sequence to read a LOB into a memory buffer

Following is a sample code sequence to read a 1024-byte LOB from a memory buffer into an Oracle 8 LOB parameter as part of an INSERT statement. Note that QALoad will create a temporary memory buffer and will populate it with meaningless data. Comments are added to commands where appropriate.

```
DO_OCIHandleAlloc( HNDL(0), HNDL(5), OCI_HTYPE_STMT);

/* A special descriptor (often referred to as a lob locator) must be created for the lob
object. Note that the DESCRIPTOR_COUNT value in the script will need to be incremented by 1
and that the 2nd parameter in the call to DO_OCIDescriptorAlloc is DESC(n) where n is the
previous value in DESCRIPTOR_COUNT */
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);

/* DO_OCIStmtPrepare( HNDL(5), "INSERT INTO CLBTAB VALUES ( 'Test', " "EMPTY_CLOB())",
OCI_NTV_SYNTAX );

/* Note that since the LOB is empty, there is no bind call before the execute call. */
DO_OCIExecute( HNDL(5), 1, OCI_DEFAULT );

/* Use the DO_OCILobWrite call to write the LOB data. Note that 1024 bytes are being written
to the LOB column of the inserted record. */
DO_OCILobWrite(HNDL(3), HNDL(1), DESC(0), 1024, 1, 1024, 0, 0, 1 );

/* Once the LOB is written, the descriptor is freed with a call to DO_OCIDescriptorAlloc */
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
DO_OCIHandleFree( HNDL(5), OCI_HTYPE_STMT);
```

Command sequence to write a LOB from a memory buffer

Below is a sample code sequence to write a 1024-byte LOB from an Oracle 8 LOB to a memory buffer. Note that QALoad will create a temporary memory buffer to store the data. Comments are added to commands where appropriate.

```
DO_OCIHandleAlloc( HNDL(0), HNDL(6), OCI_HTYPE_STMT);

/* A special descriptor (often referred to as a lob locator) must be created for the lob
object. Note that the DESCRIPTOR_COUNT value in the script will need to be incremented by 1
and that the 2nd parameter in the call to DO_OCIDescriptorAlloc is DESC(n) where n is the
```

```
previous value in DESCRIPTOR_COUNT */
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCIStmtPrepare( HNDL(6), "SELECT essay FROM CLBTAB WHERE name = 'Test' " "for update",
OCI_NTV_SYNTAX );

/* Since the LOB is an output from the SELECT statement, a DO_OCIDefine call must associate
the select parameter with the allocated descriptor. See DO_OCIDefine for more information.
*/
DO_OCIDefine(HNDL(6), HNDL(1), 1, 1, SQLT_CLOB, 0, DESC(0));
DO_OCIExecute( HNDL(6), 1, OCI_DEFAULT );

/* Use the DO_OCILobRead call to read the LOB data from the database. Note that 1024 bytes
are being read from the LOB column of the fetched record. */
DO_OCILobRead(HNDL(3), HNDL(1), DESC(0), 1000, 1, 1024, 0, 1 );

/* Once the LOB is read, the descriptor is freed with a call to DO_OCIDescriptorAlloc */
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
DO_OCIHandleFree( HNDL(6), OCI_HTYPE_STMT);
```

DO_OCI8BindDate

Binds a date variable (created by DO_makedate) to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:).

DO_OCI8BindDate commands must be placed between the DO_OCIStmtPrepare command and the DO_OCIStmtExecute command. To bind by position, instead of by name, precede the position bind variable with an "@" symbol.

DO_OCI8BindDate is a deprecated command. It is recommended that you use DO_OCIBind instead.

Syntax

```
DO_OCI8BindDate( int statementHandleIndex, text* BindVariableName, ub1* ORADATEStructPtr );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCIStmtPrepare.
BindVariableName	The name of the bind variable as a character string.
&ORADATEStructPtr	A pointer to an ORADATE structure, define in Oracle's header files.

Equivalent OCI

```
OCIBindByName, OCIBindByPos
```

Example

The following example shows the fetch loop to retrieve data after a SQL statement is executed.

```
DO_OCI8BindDate( HNDL(5), ":CURDATE", &CURDATE1);
```

DO_OCI8BindNull

Binds a NULL value to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_OCI8BindString commands must be placed between the DO_OCIStmtPrepare command and the DO_OCIStmtExecute command. To bind by position instead of by name, precede the position bind variable with an "@' symbol.

Syntax

```
DO_OCI8BindNull(int statementHandleIndex, text* BindVariableName );
```

Return Value

Parameters

Parameter	Description
`statementHandleIndex`	`An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCIStmtPrepare.`
`BindVariableName`	`The name of the bind variable as a null-terminated character string.`

Equivalent OCI

```
OCIBindByName, OCIBindByPos
```

Example

The following example shows the :DUMMY bind variable being bound to a NULL value.

```
DO_OCI8BindNull(HNDL(5), ":DUMMY");
```

DO_OCI8BindString

Binds a program variable to a bind variable in an SQL statement.

Bind variables are specified in SQL statements by preceding the variable name with a colon (:). DO_OCI8BindString commands must be placed between the DO_OCIStmtPrepare command and the DO_OCIStmtExecute command. To bind by position instead of by name, precede the position bind variable with an "@' symbol.

DO_OCI8BindString only supports the binding of strings, nulls, or dates. If you need to bind a numeric value, convert it first to a string before passing it to DO_OCI8BindString. If needed, Oracle automatically

converts character data types to numeric.

DO_OCI8BindString is a deprecated command. It is recommended that you use DO_OCIBind instead.

DO_OCI8BindString binds every data type as a fixed character and forces the Oracle server to make implicit database conversions. Also, you must variablize OUTPUT variables or they will overwrite the input data held by string constants.

Syntax

```
DO_OCI8BindString( int statementHandleIndex, text* BindVariableName, ub1* ValueString );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous call to OCIStmtPrepare.
BindVariableName	The name of the bind variable as a null-terminated character string.
ValueString	A pointer to a string containing the value for the bind variable (null terminated).

Equivalent OCI

```
OCIBindByName, OCIBindByPos
```

Example

The following example shows the program variable CITYNAME being bound to the :CITY bind variable.

```
DO_OCI8BindString( HNDL(5), ":CITY", CITYNAME );
```

DO_OCI8GetSelectData

Copies the data retrieved from an Oracle8 SQL SELECT statement into a program variable.

DO_OCI8GetSelectData processes the data retrieved by DO_OCIProcessSelectList() by copying the value of the fetched data to another program variable. Typically, the program variable is also a source variable. Source variables are created by ActiveData for Oracle so that postbind and/or fetch data from one portion of a script can be used as input to subsequent bind statements.

Note: If you are working with Oracle 7 select output data, use DO_GetSelectData instead.

If the formatType is INT_FORMAT, then the data is converted to an integer before formatting (using atol()).

This implies that the formatString contains a %i, %d or equivalent.

Syntax

```
DO_OCI8GetSelectData( int fetchCount, int colnum, int rowNum, char** srcName,
OracleFormatDataTypeEnum formatType, char* formatString, int addConstant);
```

Return Value

Parameters

Parameter	Description
fetchCount	A number from 1-n indicating which fetch sequence to use to fetch the data. The script code for a fetch statement is generally output as a C-based while-loop. This loop retrieves data until no more data is available. This parameter tells which iteration of that loop to use to retrieve the data.
colnum	Column number to use to fetch the data. The first column is 1.
rowNum	Row number to use to fetch the data. The first row number is 1.
srcName	Pointer to the address of a source variable. The function allocates memory for the source value and copy its value into this variable. Note that this parameter is a char**.
formatType	*OracleFormatDataTypeEnum* Data type to be used in the special format string. Acceptable values are: Value — Description _NONE_ — No special formatting. INT_FORMAT — Convert bind data to an integer before formatting. STRING_FORMAT — Assume that the bind data is not numeric.
formatString	A printf-style format. The data is formatted using this string. Only used if the formatType is INT_FORMAT or STRING_FORMAT.
addConstant	If the formatType is INT_FORMAT, this value is added to the value of the fetch data before conversion.

Example

The following example copies the fetched value of the first select-list item of the second row (in the first fetch iteration which retrieves 409 rows) to program variable FD_stmnt_3_col_1_row_2.
It also copies the fetched value of the fourth select-list item of the seventh row (in the first fetch iteration) to program variable FD_stmnt_3_col_4_row_7.

```
DO_OCIStmtExecute( HNDL(6)); /* Exec for statement 3 */

while (DO_OCIProcessSelectList (HNDL(6), 409) )
{
DO_OCI8GetSelectData (FETCH(1), COL(1), ROW(2),
               &FD_stmnt_3_col_1_row_2,
               _NONE_, "", 0);
DO_OCI8GetSelectData(FETCH(1), COL(4), ROW(7),
               &FD_stmnt_3_col_4_row_7,
               INT_FORMAT, "%04i", ADD(3));
```

```
DO_OCI8GetSelectData (FETCH(1), COL(5), ROW(2),
               &FD_stmnt_3_col_5_row_2,
               _NONE_, "", 0);
} /* end of DO_OCIProcessSelectList */
```

DO_OCI8InitIndp

An OCI8-specific routine that initializes the pointer of the null indicator variable, which is a parameter of the DO_OCIBind.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function should not be moved or modified.

Syntax

```
DO_OCI8InitIndp( makeOCI8Indp, pOCI8Indp, OCI8_INDP_COUNT );
```

Return Value

Parameters

Parameter	Description
makeOCI8Indp	A pointer to an integer array that holds the values of the null indicator variable.
pOCI8Indp	A pointer to makeOCI8Indp. Each element holds the pointer to the corresponding null indicator variable. In the example pOCI8Indp [0] = & makeOCI8Indp[0], the contents of makeOCI8Indp[0] is assigned before the call to DO_OCIBind.
OCI8_INDP_COUNT	The number of indicator variables utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define OCI8_INDP_COUNT at the beginning of the script. Every bind does not necessarily utilize a pOCI8Indp. If the null indicator was recorded as NULL, NULL replaces the use of pOCI8Indp.

Example

```
#define OCI8_INDP_COUNT 20
:
:
sb2* pOCI8Indp [OCI8_INDP_COUNT]; /* sb2 is a signed integer
:
sb2 makeOCI8Indp [OCI8_INDP_COUNT];
:
:
DO_OCI8InitIndp( makeOCI8Indp, pOCI8Indp, OCI8_INDP_COUNT );
:
:
makeOCI8Indp[0]=0;
makeOCI8Alen[0]=4;
DO_OCIBind( HNDL(9), HNDL(2), "@2", _INTEGER, 4,
       pOCI8Alen[0], pOCI8Indp[0], (ub1 *) "0",
       (ub1 *) &INTEGER_9_2_0 );
```

DO_OCI8InitAlen

dex

An Oracle 8-specific routine that initializes the pointer to the variable representing the length of data that is a parameter in DO_OCIBind.

This is a required initialization routine that is inserted into a script when it is generated and called before synchronization. This function is not always called. For example, a script may not contain any DO_OCIBind calls, or the bind calls that are contained in the script do not utilize pOCI8Alen. This function should not be moved or modified.

Syntax

```
DO_OCI8InitAlen( makeOCI8Alen, pOCI8Alen, OCI8_ALEN_COUNT );
```

Return Value

Parameters

Parameter	Description
makeOCI8Alen	A pointer to an array that holds the values of the length of data.
pOCI8Alen	A pointer to makeOCI8Alen. Each element holds the pointer to the corresponding makeOCI8Alen. In the example, pOCI8Alen [0] = & makeOCI8Alen [0], the contents of makeOCI8Alen [0] are assigned before the call to DO_OCIBind.
OCI8_ALEN_COUNT	The number of pOCI8Alen utilized in the script. If this number is incorrect, the script will fail. The number can be modified in the #define OCI8_ALEN_COUNT at the beginning of the script. Every bind does not necessarily utilize a pOCI8Alen. For example, if the alen was captured as NULL, NULL replaces the use of pOCI8Alen.

Example

```
#define OCI8_ALEN_COUNT 20
:
:
sb2*pOCI8Alen [OCI8_ALEN_COUNT]; /* sb2 is a signed integer */
sb2 makeOCI8Alen [OCI8_ALEN_COUNT];
:
:
DO_OCI8InitAlen( makeOCI8Alen, pOCI8Alen, OCI8_ALEN_COUNT );
:
:
makeOCI8Indp[0]=0;
makeOCI8Alen[0]= 4;
DO_OCIBind( HNDL(9), HNDL(2), "@2", _INTEGER,
      4, pOCI8Alen[0], pOCI8Indp[0],
      (ub1 *) "0", (ub1 *) &INTEGER_9_2_0 );
```

DO_OCIAttrSet

Sets a particular attribute for a previously allocated Oracle 8 OCI handle.

Syntax

```
DO_OCIAttrSet( int targetHandleIndex, int targetHandleType, void* attributep, int
attributeSize, OCIAttributeTypeEnum attributeType, int errorHandleIndex, int
attributeHandleIndex );
```

Return Value

Parameters

Parameter	Description
targetHandleIndex	An index to a previously allocated Oracle handle whose attribute is to be set.
targetHandleType	The handle type.
attributep	A pointer to the attribute value. This can be a character string or another handle in select instances.
attributeSize	The size of the attribute value.
attributeType	*OCIAttributeTypeEnum* The type of attribute to set for the handle. Value — Description OCI_ATTR_USERNAME — OCI8 username attribute. OCI_ATTR_PASSWORD — OCI8 password attribute.
errorHandleIndex	An index to a previously allocated Oracle 8 error handle.
attributeHandleIndex	An index to a previously allocated Oracle 8 attribute handle.

Equivalent OCI

OCIAttrSet

Example

This example sets OCI_ATTR_USERNAME and OCI_ATTR_PASSWORD attributes of the session handle prior to calling DO_OCISessionBegin to start an Oracle8 session on a previously attached database.

```
DO_OCIAttrSet( HNDL(5), OCI_HTYPE_SESSION, "scott", 5,
OCI_ATTR_USERNAME, HNDL(1), IS_ATTRIBUTE);
DO_OCIAttrSet( HNDL(5), OCI_HTYPE_SESSION, "tiger", 5,
OCI_ATTR_PASSWORD, HNDL(1), IS_ATTRIBUTE);
```

DO_OCIBind

Binds a program variable to a bind variable in an SQL statement.

A DO_OCIBind is generated wherever a bind occurs in the capture file. Note that the binds only support single (scalar) values, not array values. DO_OCIBind accurately reproduces the original bind call made by the application. This eliminates extra data conversion steps and improves handling of OUTPUT variables in Oracle stored procedures.

Bind variables are specified in SQL statements by preceding the variable names with a colon (:). DO_OCIBind must be called after the DO_OCIStmtPrepare and before a DO_OCIStmtExecute.

Once you have bound a variable, you can change its value and length and execute it again without reparsing the SQL statement or rebinding the variable. Currently, DO_OCIBind only supports the datatypes supported by DO_ScalarBindA.

Syntax

```
DO_OCIBind( int statementHandleIndex, int errorHandleIndex, text* BindVariable,
OCIExternalDataTypeEnum DataType, sword OutputBufferLength, ub2* OCI8Alen, sb2* OCI8Indp,
ub1* InputBuffer, ub1* OutputBuffer);
```

Return Value

Parameters

Parameter	Description
`statementHandleIndex`	`An index to the current allocated Oracle 8 statement handle used in the DO_OCIStmtPrepare call.`
`errorHandleIndex`	`An index to an allocated Oracle 8 error handle.`
`BindVariable`	`A pointer to the name of the null-terminated bind variable string.`
`DataType`	*OCIExternalDataTypeEnum* External datatype of the bind variable. Valid Oracle external datatypes (with program variable types) include: Value — Description SQLT_CHR — C Datatype:(char[n]) SQLT_NUM — C Datatype:(unsigned char[21]) SQLT_INT — C Datatype:(signed char) SQLT_FLT — C Datatype:(float, double) SQLT_STR — C Datatype:(char[n+1]) SQLT_VNU — C Datatype:(char[22]) SQLT_LNG — C Datatype:(char[n])

	SQLT_VCS	C Datatype:(char[n] + sizeof(short int)])
	SQLT_DAT	C Datatype:(char[7])
	SQLT_VBI	C Datatype:(unsigned char[n + sizeof(short int)]
	SQLT_BIN	C Datatype:(unsigned char[n])
	SQLT_LBI	C Datatype:(unsigned char[n])
	SQLT_UIN	C Datatype:(unsigned)
	SQLT_LVC	C Datatype:(char[n + sizeof(int)])
	SQLT_LVB	C Datatype:(unsigned char[n + sizeof(int)])
	SQLT_AFC	C Datatype:(char[n])
	SQLT_AVC	C Datatype:(char[n + 1)
	SQLT_CLOB	Character (ASCII) LOB
	SQLT_BLOB	Binary LOB
	SQLT_FILE	File LOB
OutputBufferLength	Size of the output buffer. This is the maximum size of the OutputBuffer buffer. If binding a PL/ SQL OUTPUT variable, this value must be at least as large as the expected output variable.	
OCI8Alen	Pointer to a variable that contains the length of the bind data. This is an alternative method of defining the length of the output data. Use the makeOCI8Alen macro to create this pointer. OCI8Alen should only be used if it is necessary to determine the length of the bind value returned from a statement execute. For character strings, using the strlen on the OutputBuffer variable after the statement execute is an easier method of obtaining this length.	
OCI8Indp	Pointer to an indicator that the bind variable is NULL. Use the makeOCI8Indp macro to create this pointer.Using the DO_OCIBindNull call is a preferred way of binding a NULL value unless the bind variable is aPL/SQL OUTPUT variable.	
InputBuffer	Pointer to a buffer containing the input data.	
OutputBuffer	Pointer to the output data buffer.	

Equivalent OCI

OCIBindByName, OCIBindByPos

Example

```
DO_OCIBind(HNDL(5), HNDL(1), ":PKEY", _VARCHAR2, strlen(PB_PKEY), NULL, NULL, (ub1 *)
PB_PKEY, (ub1 *) VARCHAR2_6_PKEY_1);
```

DO_OCICommit

Commits the current Oracle8 transaction. A commit should be performed after all relevant SQL statements have been processed.

DO_OCICommit is a deprecated command. It is recommended that you use DO_OCITransCommit instead.

Note: DO_OCICommit should only be used in a single-user environment. For multi-user environments, use DO_OCITransCommit.

Syntax

```
DO_OCICommit(int errorHandleIndex, ub4 CommitType);
```

Return Value

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to commit. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for *QA*Load scripts.

Equivalent OCI

```
OCITransCommit
```

Example

The following example shows a SQL statement being committed after the execute and fetch loop.

```
DO_OCIStmtExecute( HNDL(5) ); /* Exec for statement 3 */
DO_OCICommit( HNDL(5), HNDL(1), OCI_DEFAULT );
```

DO_OCIDefine

Associates an item in a select-list to an Oracle external datatype and an output data buffer.

Syntax

```
DO_OCIDefine( int statementHandleIndex, int errorHandleIndex, ub4 fetchcount, ub4
SelectListPosition, ub4 DataType, sb4 BufferLength, int lobDescriptorIndex );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously

	used in the call to DO_OCIStmtPrepare.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
FetchCount	The fetch count as defined in DO_OCIProcessSelectList. This value should be set to the value defined to DO_OCIProcessSelectList or to 1.
selectListPosition	The position of the item in the select list. The starting point is 1.
DataType	The Oracle external datatype.
BufferLength	The maximum data length for the output buffer for the defined value.
lobDescriptorIndex	An index to an previously allocated Oracle 8 lob locator, if the output is a BLOB, CLOB, or BFILE. If a lob locator is not used, the value is IS_ATTRIBUTE.

Equivalent OCI

OCIDefineByPos

Example

The following example shows the select-list item EMPNO being defined as having position 1 and a string type.

```
DO_OCIStmtPrepare( HNDL(5), "SELECT EMPNO FROM EMP", OCI_NTV_SYNTAX );
:
:
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _STRING, 33, IS_ATTRIBUTE);
```

DO_OCIDescriptorAlloc

Allocates and initializes an Oracle 8 OCI descriptor or LOB locator.

Syntax

```
DO_OCIDescriptorAlloc( int parentHandleIndex, int descriptorIndex, OCIDescriptorTypeEnum
descriptorType );
```

Return Value

Parameters

Parameter	Description
parentHandleIndex	An index to an allocated Oracle 8 environment handle used as the parent handle in this call.
descriptorIndex	An index to an Oracle descriptor to be allocated and initialized.
descriptorType	*OCIDescriptorTypeEnum*

	The descriptor type. The Oracle 8 descriptor types used by QALoad commands are:
	Value — Description
	OCI_DTYPE_LOB — OCI8 LOB Descriptor.
	OCI_DTYPE_BFILE — OCI8 BFILE Descriptor.
	OCI_DTYPE_ROWID — OCI8 ROWID Descriptor.
	Equivalent OCI
	`OCIDescriptorAlloc`
	Example
	`DO_OCIDescriptorAlloc(HNDL(0), DESC(0), OCI_DTYPE_LOB);`

DO_OCIDescriptorFree

De-allocates an Oracle 8 OCI descriptor or LOB locator.

Syntax

```
DO_OCIDescriptorFree( int descriptorIndex, OCIDescriptorTypeEnum descriptorType );
```

Return Value

Parameters

Parameter	Description
`descriptorIndex`	`An index to an Oracle descriptor to be de-allocated.`
`descriptorType`	*OCIDescriptorTypeEnum* The descriptor type. The Oracle 8 descriptor types used by QALoad commands are: Value — Description OCI_DTYPE_LOB — OCI8 LOB Descriptor. OCI_DTYPE_BFILE — OCI8 BFILE Descriptor. OCI_DTYPE_ROWID — OCI8 ROWID Descriptor.

Equivalent OCI

`OCIDescriptorFree`

Example

```
DO_OCIDescriptorFree(DESC(0), OCI_DTYPE_LOB);
```

DO_OCIEnvFreeAll

De-allocates all environment handles before the end of an OCI8 script.

Syntax

```
DO_OCIEnvFreeAll();
```

Return Value

Parameters

None

Equivalent OCI

None

Example

```
DO_OCIEnvFreeAll();
:
DO_free_data();
REPORT(SUCCESS);
EXIT();
return(0);
```

DO_OCIEnvInit

Allocates and initializes an Oracle OCI8 environment handle.

Syntax

```
DO_OCIEnvInit( int envHandleIndex, int mode );
```

Return Value

Parameters

Parameter	Description
envHandleIndex	An index to the environment handle. The mode value should be set to OCI_DEFAULT.
mode	Mode for OCI8 environment initialization.

Equivalent OCI

```
OCIEnvInit
```

Example

```
DO_OCIEnvInit(HNDL(0), OCI_DEFAULT);
```

DO_OCIExecute

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCIStmtPrepare. Note that SQL syntax errors are reported at execution time.

DO_OCIExecute is a deprecated command. It is recommended that you use DO_OCIStmtExecute instead.

Syntax

```
DO_OCIExecute( int statementHandleIndex, ub4 iters, ub4 mode );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously used in the call to DO_OCIStmtPrepare.
Iterations	The number of times this statement is executed for non-SELECT statements. This value can be set to 1 for SELECT statements if, and only if, all output variables were previously defined with DO_OCIDefine. This value should be set to 1.
mode	The mode for execution. The mode value should be set to the reserved word OCI_DEFAULT.

Equivalent OCI

```
OCIStmtExecute
```

Example

```
DO_OCIExecute(HNDL(5), 1, OCI_DEFAULT);
```

DO_OCIHandleAlloc

Allocates and initializes an Oracle 8 OCI handle.

Syntax

```
DO_OCIHandleAlloc( int parentHandleIndex, int handleIndex, OCIHandleTypeEnum handleType );
```

Return Value

Parameters

Parameter	Description
parentHandleIndex	An index to an allocated Oracle 8 environment handle used as the parent handle in this call.
handleIndex	An index to an Oracle handle to be allocated and initialized.
handleType	*OCIHandleTypeEnum* The handle type. The following handle types in Oracle 8 are used by QALoad commands: Value — Description OCI_HTYPE_ERROR — OCI8 Error handle OCI_HTYPE_SVCCTX — OCI8 Service Context handle OCI_HTYPE_STMT — OCI8 Statement handle OCI_HTYPE_DESCRIBE — OCI8 Descriptor OCI_HTYPE_SERVER — OCI8 Server handle OCI_HTYPE_SESSION — OCI8 Session handle OCI_HTYPE_TRANS — OCI8 Transaction handle

Equivalent OCI

```
OCIHandleAlloc
```

Example

```
DO_OCIHandleAlloc(HNDL(0), HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCIHandleFree

De-allocates an Oracle 8 OCI handle.

Syntax

```
DO_OCIHandleFree(int handleIndex, OCIHandleTypeEnum handleType );
```

Parameters

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>parentHandleIndex</td><td>An index to an allocated Oracle 8 environment handle used as the parent handle in this call.</td></tr>
<tr><td>handleIndex</td><td>An index to an Oracle handle to be de-allocated.</td></tr>
<tr><td>handleType</td><td>OCIHandleTypeEnum
The handle type. The following handle types in Oracle 8 are used by QALoad commands:
<table><tr><th>Value</th><th>Description</th></tr><tr><td>OCI_HTYPE_ERROR</td><td>OCI8 Error handle</td></tr><tr><td>OCI_HTYPE_SVCCTX</td><td>OCI8 Service Context handle</td></tr><tr><td>OCI_HTYPE_STMT</td><td>OCI8 Statement handle</td></tr><tr><td>OCI_HTYPE_DESCRIBE</td><td>OCI8 Descriptor</td></tr><tr><td>OCI_HTYPE_SERVER</td><td>OCI8 Server handle</td></tr><tr><td>OCI_HTYPE_SESSION</td><td>OCI8 Session handle</td></tr><tr><td>OCI_HTYPE_TRANS</td><td>OCI8 Transaction handle</td></tr></table></td></tr>
</table>

Equivalent OCI

```
OCIHandleAlloc
```

Example

```
DO_OCIHandleFree(HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCIInitialize

Initializes the Oracle OCI8 process environment.

This command must be issued once in a QALoad script prior to any other Oracle8 script commands, and should be outside any QALoad transactions.

Syntax

```
DO_OCIInitialize( int mode );
```

Return Value

Parameters

Parameter	Description
mode	OCI8 process environment mode. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCIInitialize

Example

```
DO_OCIInitialize(OCI_DEFAULT);
```

DO_OCILdaToSvcCtx

Toggles an Oracle 7 logon data area to an Oracle 8 service context handle.

This should be done after using DO_OCISvcCtxToLda to create Oracle 7 in a database session in Oracle 8.

Syntax

```
DO_OCILdaToSvcCtx( int svcContextHandleIndex, int errorHandleIndex, int LdaIndex );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
LdaIndex	An index to a Logon Data area.

Equivalent OCI

OCILdaToSvcCtx

Example

The following example shows using the Oracle7 logon data area (LDA) to create an Oracle8 service context handle using the DO_OCILdaToSvcCtx call.

```
DO_OCILdaToSvcCtx ( HNDL(4), HNDL(2), LDA(0) );
```

DO_OCILobRead

Reads a LOB into a buffer.

Syntax

```
DO_OCILobRead( int svcContextHandleIndex, int errorHandleIndex, int lobDescriptorIndex, ub4 ReadCount, ub4 LOBOffset, ub4 BufferLength, ub2 CharSetID, ub1 CharSetFrm );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
lobDescriptorIndex	An index to an Oracle 8 LOB locator previously allocated with a DO_OCIDescriptorAlloc call.
ReadCount	On input, the number of characters (for CLOB) or bytes (for BLOB) to be read. This variable contains the actual number of bytes or characters read after the call.
LOBOffset	On input, the absolute offset from the beginning of the LOB file. For CLOBs, this is the number of characters from the beginning. For BLOBs, it is the numbers of bytes. The first position is 1.
BufferLength	The length of the buffer. This value is specified in bytes.
CharSetID	The character set ID of the buffer data.
CharSetFrm	The character set form of the buffer data.

Equivalent OCI

OCILobRead

Example

The following example will perform a LOB read of 1024 bytes from the database.

```
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCIStmtPrepare( HNDL(5), "SELECT essay FROM CLBTAB WHERE name = 'Test' " "for update",
OCI_NTV_SYNTAX );
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _CLOB, 0, DESC(0));
DO_OCIExecute( HNDL(5), 1, OCI_DEFAULT );
DO_OCILobRead(HNDL(3), HNDL(1), DESC(0), 1000, 1, 1024, 0, 1 );
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
```

DO_OCILobWrite

Writes the contents of a buffer into an Oracle 8 LOB.

Syntax

```
DO_OCILobWrite( int svcContextHandleIndex, int errorHandleIndex, int lobDescriptorIndex, ub4
ReadCount, ub4 LOBOffset, ub4 BufferLength, ub1 LOBPiece, ub1 CharSetID, ub2 CharSetFrm );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to the current allocated Oracle 8 service context handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
lobDescriptorIndex	An index to an Oracle 8 LOB locator previously allocated with a DO_OCIDescriptorAlloc call.
ReadCount	On input, the number of characters (for CLOB) or bytes (for BLOB) to be written. This variable contains the actual number of bytes or characters written after the call.
LOBOffset	On input, the absolute offset from the beginning of the LOB file. For CLOBs,this is the number of characters from the beginning. For BLOBs, it is the numbers of bytes. The first position is 1.
BufferLength	The length of the buffer. This value is specified in bytes.
LOBPiece	The piece of the LOB buffer being written.
CharSetID	The LOB character set ID of the buffer data.
CharSetFrm	The LOB Character set form of the buffer data.

Equivalent OCI

```
OCILobWrite
```

Example

The following example will perform a LOB write of 1024 bytes to the database.

```
DO_OCIDescriptorAlloc( HNDL(0), DESC(0), OCI_DTYPE_LOB);
DO_OCIStmtPrepare( HNDL(5), "INSERT INTO CLBTAB VALUES ( 'Jack', " "EMPTY_CLOB())",
OCI_NTV_SYNTAX );
DO_OCIDefine(HNDL(5), HNDL(1), 1, 1, _CLOB, 0, DESC(0));
DO_OCIExecute( HNDL(5), 1, OCI_DEFAULT );
DO_OCILobWrite(HNDL(3),HNDL(1),DESC(0), 1024, 1, 1024, 0, 0, 1 );
DO_OCIDescriptorFree( HNDL(0), OCI_DTYPE_LOB);
```

DO_OCILogoff

Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

DO_OCILogoff is a deprecated command. It is recommended that you use DO_OCILogoffEx instead.

Note: DO_OCILogoff logs off the most recent Oracle logon in the QALoad script. When using DO_OCILogon/DO_OCILogoff, make sure that there are no overlapping sessions.

Syntax

```
DO_OCILogoff( int errHandleIndex );
```

Note: svcContextHandleIndex is not a parameter to DO_OCILogoff. If more than one Oracle Logon is in the script, subsequent logons should be logged off with DO_OCILogoffEx.

Return Value

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.

Equivalent OCI

```
OCILogoff
```

Example

```
DO_OCILogoff(HNDL(1));
```

DO_OCILogoffEx

Terminates an Oracle OCI8 logon session and connection created with DO_OCILogon.

Note: DO_OCILogoffEx will log off the Oracle logon in the QALoad script. When the script is using DO_OCILogon/DO_OCILogoffEx, QALoad playback uses OCIServerAttach, OCISessionBegin, OCIServerDetach, and OCISessionEnd calls to prevent threading issues. OCILogon and OCILogofff calls are not thread-safe.

Syntax

```
DO_OCILogoffEx( int svcContextHandleIndex, int errHandleIndex );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.

Equivalent OCI

```
OCILogoff
```

Example

```
DO_OCILogoffEx (HNDL(3), HNDL(1));
```

DO_OCILogon

Creates a simple Oracle OCI8 logon connection and session for *QA*Load . Any application must log on to Oracle before performing any other Oracle operations.

For DO_OCILogon, three components must be provided:

- User's login ID
- User's password
- Database name as recognized by Oracle Net8 software.

Note: When the script is using DO_OCILogon/DO_OCILogoffEx, QALoad uses OCIServerAttach, OCISessionBegin, OCIServerDetach, and OCISessionEnd calls to prevent threading issues. OCILogon and OCILogoff calls are not thread-safe.

Syntax

```
DO_OCILogon( int envHandleIndex, int errHandleIndex, int svcContextHandleIndex, text*
username, ub4 uname_len, text* password, ub4 passwd, text* dbname, ub4 dbname_len );
```

Return Value

Parameters

Parameter	Description
envHandleIndex	An index to the environment handle.
errHandleIndex	An index to an allocated Oracle 8 error handle.
svcContextHandleIndex	An index to an Oracle service context handle index. This handle is automatically allocated by this call, and is automatically deallocated by a DO_OCILogoffEx call.
username	Oracle 8 user login ID.
uname_len	Character length of user login ID.
password	Oracle 8 user password for login ID.
passwd	Character length of user password.
dbname	Name of the data source to connect to.
dbname_len	Character length of data source name.

Equivalent OCI

```
OCILogon
```

Example

```
DO_OCILogon(HNDL(0), HNDL(1), "scott", 5, "tiger", 5, "oradb.world", 11);
```

DO_OCIProcessSelectList

Fetches select-list data from an Oracle 8 database after an OCIStmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

If there are no DO_OCIDefine calls before the DO_OCIStmtExecute call for the select statement, the call builds up a set of internal buffers to store the returned data (otherwise done by DO_OCIDefine calls). All data is returned as ASCII strings.

Syntax

```
DO_OCIProcessSelectList( int statementHandleIndex, int fetchcount );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous calls to OCIStmtPrepare and OCIStmtExecute.
fetchCount	The count of rows to be returned per fetch loop iteration. The FetchCount value should be set to 1 unless the exact count of fetched rows is known. Note that the loop iterates until there are no more rows satisfying the SQL select request, not when the fetchCount value is reached.

Equivalent OCI

```
OCIStmtFetch
```

Example

The following example shows the DO_OCIProcessSelectList () fetch loop retrieving data after a SQL statement is executed. Note that only 1 row is fetched per loop iteration. In addition, the fetched value is processed by DO_OCI8GetSelectData().

```
DO_OCIStmtExecute( HNDL(0), HNDL(5), HNDL(1), 1, OCI_DEFAULT );
while ( DO_OCIProcessSelectList(HNDL(5), 1 ) )
{
DO_OCI8GetSelectData( FETCH(1),COL(1), ROW(1), &FD_stmnt_4_col_1_row_1, _NONE_, "", 0 );
}
```

DO_OCIProcessSelectList_EX

Fetches select-list data from an Oracle 8 database after an OCIStmtExecute call. It is called repeatedly in a loop until there are no more rows satisfying the SQL select request.

If there are no DO_OCIDefine calls before the DO_OCIStmtExecute call for the select statement, the will builds up a set of internal buffers to store the returned data (otherwise done by DO_OCIDefine calls). All data is returned as ASCII strings.

DO_OCIProcessSelectList_EX extends the DO_OCIProcessSelectList macro by accommodating nested OCI8 logins. Beginning with QALoad 5.0, Compuware recommends that you use DO_OCIProcessSelectList_EX.

Syntax

```
DO_OCIProcessSelectList_EX( int statementHandleIndex, int errorHandleIndex, int fetchcount
);
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle used as the handle in the previous calls to OCIStmtPrepare and OCIStmtExecute.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
fetchCount	The count of rows to be returned per fetch loop iteration. The FetchCount value should be set to 1 unless the exact count of fetched rows is known. Note that the loop iterates until there are no more rows satisfying the SQL select request, not when the fetchCount value is reached.

Equivalent OCI

```
OCIStmtFetch
```

Example

The following example shows the fetch loop retrieving data after a SQL statement is executed.

```
while (DO_OCIProcessSelectList_EX( HNDL(5), HNDL(2), 1 ) );

{
} /*end of DO_process_select_list */
```

DO_OCIRollback

Rolls back the current Oracle8 transaction.

DO_OCIRollback is a deprecated command. It is recommended that you use DO_OCITransRollback instead.

Note: DO_OCIRollback should only be used in a single-user environment. For multi-user environments, use DO_OCITransRollback.

Syntax

```
DO_OCIRollback ( int errorHandleIndex, ub4 CommitType );
```

Return Value

Parameters

Parameter	Description
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to roll back. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for *QA*Load scripts.

Equivalent OCI

```
OCITransRollback
```

Example

The following example shows a SQL statement being rolled back after the execute.

```
DO_OCIStmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" ", :5 )",
OCI_NTV_SYNTAX );
:
:
:
DO_OCIStmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCIRollback( HNDL(1), OCI_DEFAULT );
```

DO_OCIServerAttach

Creates a standard Oracle OCI8 database connection for *QA*Load . Note that individual Oracle 8 user logons are done with the DO_OCIServerAttach command.

Any application must log on to Oracle before performing any other Oracle operations.
For DO_OCIServerAttach, the connect string for a database (dblink parameter) must be provided.

Syntax

```
DO_OCIServerAttach( int serverHandleIndex, int errHandleIndex, text* dblink, sb4 dblink_len,
int mode );
```

Return Value

Parameters

Parameter	Description
serverHandleIndex	An index to an allocated Oracle 8 service handle.
errorHandleIndex	An index to an allocated Oracle 8 error handle.

dblink	Name of the data source to connect to.
dblink_len	Character length of database name.
mode	The mode of operation.

Equivalent OCI

OCIServerAttach

Example

This example shows the commands necessary to attach to an Oracle8 database.

```
DO_OCIHandleAlloc( HNDL(0), HNDL(2), OCI_HTYPE_SERVER);
DO_OCIHandleAlloc( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);
DO_OCIServerAttach(HNDL(2), HNDL(1), "oradb.world", 11, OCI_DEFAULT);
```

DO_OCIServerDetach

Detaches *QA*Load from the Oracle OCI8 data source connection previously attached to with the DO_OCIServerAttach command.

Note that all users must be logged off with the DO_OCISessionEnd command before this call.

Syntax

```
DO_OCIServerDetach( int serverHandleIndex, int errHandleIndex, int mode );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle previously used in a call to DO_OCIServerAttach.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
mode	Mode of operation for Oracle 8 session. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

OCIServerDetach

Example

This command shows the process of detaching from an Oracle 8 server and freeing the respective handles.

```
DO_OCIServerDetach(HNDL(2), HNDL(1), OCI_DEFAULT);
DO_OCIHandleFree( HNDL(2), OCI_HTYPE_SERVER);
DO_OCIHandleFree( HNDL(1), OCI_HTYPE_ERROR);
```

DO_OCISessionBegin

Creates an Oracle OCI8 logon session for *QA*Load to a server previously attached to with DO_OCIServerAttach.

Any application must log on to Oracle before performing any other Oracle operations.

Syntax

```
DO_OCISessionBegin( int svcContextHandleIndex, int errHandleIndex, int sessionHandleIndex,
int credt, int mode );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle used previously in DO_OCIServerAttach.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
sessionHandleIndex	An index to an allocated Oracle 8 session handle.
credt	Credentials for attachment to Oracle server. The Credentials value should be set to OCI_CRED_RDBMS if the username and password are required to log into the Oracle 8 database. If the database uses integrated security, set Credentials to OCI_CRED_EXT.
mode	Mode of operation for Oracle 8 session. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

```
OCISessionBegin
```

Example

This example shows the commands needed to begin a user session on an Oracle 8 database that has been previously attached by DO_OCIServerAttach.

```
DO_OCIHandleAlloc( HNDL(0), HNDL(3), OCI_HTYPE_SVCCTX);
DO_OCIAttrSet( HNDL(3), OCI_HTYPE_SVCCTX, 0, 0, OCI_ATTR_SERVER, HNDL(1), HNDL(2));
DO_OCIHandleAlloc( HNDL(0), HNDL(4), OCI_HTYPE_SESSION);
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),
IS_ATTRIBUTE);
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),
IS_ATTRIBUTE);
DO_OCISessionBegin(HNDL(3), HNDL(1), HNDL(4), OCI_CRED_RDBMS, OCI_DEFAULT);
```

DO_OCISessionEnd

Terminates an Oracle user session previously created with the DO_OCISessionBegin command.

Syntax

```
DO_OCISessionEnd( int svcContextHandleIndex, int errorHandleIndex, int sessionHandleIndex,
int mode );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to an allocated Oracle 8 service context handle previously used in the call to DO_OCISessionBegin.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
sessionHandleIndex	An index to an allocated Oracle 8 session handle previously used in the call to DO_OCISessionBegin.
mode	The mode of operation. The mode value should be set to OCI_DEFAULT.

Equivalent OCI

```
OCISessionEnd
```

Example

```
In the following example, the session logged on with the user name Scott and password tiger
is terminated by DO_OCISessionEnd.
```

```
DO_OCIHandleAlloc( HNDL(0), HNDL(4), OCI_HTYPE_SESSION );
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "scott", 5, OCI_ATTR_USERNAME, HNDL(1),
IS_ATTRIBUTE );
DO_OCIAttrSet( HNDL(4), OCI_HTYPE_SESSION, "tiger", 5, OCI_ATTR_PASSWORD, HNDL(1),
IS_ATTRIBUTE );
:
:
DO_OCISessionBegin( HNDL(3), HNDL(1), HNDL(4),
OCI_CRED_RDBMS, OCI_DEFAULT );
DO_OCISessionEnd( HNDL(3), HNDL(1), HNDL(4), OCI_DEFAULT );
DO_OCIHandleFree( HNDL(3), OCI_HTYPE_SVCCTX );
DO_OCIHandleFree( HNDL(4), OCI_HTYPE_SESSION );
```

DO_OCIStmtExecute

Executes the SQL statement or a PL/SQL block previously associated with the Oracle 8 statement handle with DO_OCIStmtPrepare. Note that SQL syntax errors are reported at execution time.

Note: In multi-user environments, use this statement in place of DO_OCIExecute.

Syntax

```
DO_OCIStmtExecute ( int svcContextHandleIndex, int statementHandleIndex, int
errorHandleIndex, ub4 iters, ub4 mode );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
statementHandleIndex	An index to an allocated Oracle 8 statement handle previously used in the call to DO_OCIStmtPrepare.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
Iterations	The number of times this statement is executed for non-SELECT statements. This value can be set to 1 for SELECT statements if, and only if, all output variables were previously defined with DO_OCIDefine. This value should be set to 1.
mode	The mode for execution. The mode value should be set to the reserved word OCI_DEFAULT.

Equivalent OCI

OCIStmtExecute

Example

```
DO_OCIStmtExecute( HNDL (3), HNDL (5),HNDL (1), OCI_DEFAULT );
```

DO_OCIStmtPrepare

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

Oracle 8 SQL statements are not sent to the Oracle 8 server until execution time (handled by QALoad command DO_OCIStmtExecute). SQL syntax errors are reported at execution time.

Syntax

```
DO_OCIStmtPrepare( int statementHandleIndex, text* SQLStatement, ub4 OracleSyntax );
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle.
SQLStatement	A pointer to a null-terminated string containing the SQL

	statement.
OracleSyntax	A variable flag for the parsing syntax. This value should be OCI_NTV_SYNTAX. The value for OracleLanguage should be set to OCI_NTV_SYNTAX (which defers the parsing syntax to the Oracle database) unless you want to specify a parsing syntax explicitly. Other possible values are OCI_V7_SYNTAX and OCI_V8_SYNTAX for specifying a parsing syntax based on Oracle 7 and Oracle 8, respectively.

Equivalent OCI

OCIStmtPrepare

Example

The following example shows the preparation of a typical SQL statement.

```
DO_OCIStmtPrepare(HNDL(5), "SELECT * FROM EMP;", OCI_NTV_SYNTAX);
```

DO_OCIStmtPrepare_EX

Prepares a SQL statement or a PL/SQL block and associates it with an Oracle 8 statement handle.

Oracle 8 SQL statements are not sent to the Oracle 8 server until execution time (handled by QALoad command DO_OCIStmtExecute). SQL syntax errors are reported at execution time.

DO_OCIStmtPrepare_EX extends DO_OCIStmtPrepare macro by accommodating nested OCI8 logins. Starting with QALoad 5.0, Compuware recommends that you use DO_OCIStmtPrepare_EX.

Syntax

```
DO_OCIStmtPrepare_EX( int statementHandleIndex, text* SQLStatement, int errorHandleIndex,
ub4 OracleSyntax);
```

Return Value

Parameters

Parameter	Description
statementHandleIndex	An index to an allocated Oracle 8 statement handle.
SQLStatement	A pointer to a null-terminated string containing the SQL statement.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
OracleSyntax	A variable flag for the parsing syntax. This value should be OCI_NTV_SYNTAX. The value for OracleLanguage should be set to OCI_NTV_SYNTAX (which defers the parsing syntax to the Oracle database) unless you want to specify a parsing syntax explicitly. Other possible values are OCI_V7_SYNTAX and OCI_V8_SYNTAX for specifying a parsing syntax based on Oracle 7 and Oracle 8, respectively.

Equivalent OCI

`OCIStmtPrepare`

Example

The following example shows the preparation of a typical SQL statement.

```
DO_OCIStmtPrepare_EX(HNDL(5), "SELECT * FROM EMP;", HNDL(2), OCI_NTV_SYNTAX );
```

DO_OCISvcCtxToLda

Toggles an Oracle 8 service context handle to an Oracle 7 logon data area. This allows Oracle 7 cursors to be created in a database session created in Oracle 8.

Syntax

```
DO_OCISvcCtxToLda( int svcContextHandleIndex, int errorHandleIndex, int LdaIndex );
```

Return Value

Parameters

Parameter	Description
`svcContextHandleIndex`	An index to the current allocated Oracle 8 service context handle.
`errorHandleIndex`	An index to an allocated Oracle 8 error handle.
`LdaIndex`	An index to a Logon Data area.

Equivalent OCI

`OCISvcCtxToLda`

Example

The following example shows toggling the Oracle8 service context handle to an Oracle7 logon data area (LDA) using the DO_OCISvcCtxToLda call.

```
DO_OCISvcCtxToLda(HNDL(4), HNDL(2), LDA(0));
```

DO_OCITransCommit

Commits the current Oracle 8 transaction. A commit should be performed after all relevant SQL statements have been processed.

Syntax

```
DO_OCITransCommit ( int svcContextHandleIndex, int errorHandleIndex, ub4 CommitType );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to commit. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for *QA*Load scripts.

Example

The following example shows an insert transaction being committed after the execute.

```
DO_OCIStmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" ", :5 )",
OCI_NTV_SYNTAX );
:
:
:
DO_OCIStmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCITransCommit ( HNDL(3), HNDL(1), OCI_DEFAULT );
```

DO_OCITransRollback

Rolls back the current Oracle 8 transaction.

Syntax

```
DO_OCITransRollback ( int svcContextHandleIndex, int errorHandleIndex, ub4 CommitType );
```

Return Value

Parameters

Parameter	Description
svcContextHandleIndex	An index to a service context handle previously used in an Oracle 8 logon.
errorHandleIndex	An index to an allocated Oracle 8 error handle.
CommitType	The type of transaction to roll back. This value should be set to the Oracle 8 reserved word OCI_DEFAULT for *QA*Load scripts.

Equivalent OCI

```
OCITransRollback
```

Example

```
DO_OCIStmtPrepare( HNDL(5), "INSERT INTO MIKE.t_session ( session_key, user_key"
",login_time_stamp, session_number, session_seq ) VALUES (:1, :2, :3, :4" ", :5 )",
OCI_NTV_SYNTAX );
:
:
```

```
:
DO_OCIStmtExecute ( HNDL(3), HNDL(5), 1, OCI_DEFAULT );
DO_OCITransRollback ( HNDL(3), HNDL(1), OCI_DEFAULT );
```

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.

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

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

ofsHTTPSDoSSLHandshake
Establishes an SSL socket connection and starts an SSL handshake.

ofsHTTPSetHdrProperty
Establishes the HTTP headers to use for connecting to the Forms servlet and listener servlet.

ofsHTTPSetListenerServletParms
Sets the Forms Listener Servlet parameters prior to connection.

ofsHTTPConnectToFormsServlet
Opens an HTTP connection to the Forms servlet responsible for initiating a Forms applet instance.

ofsHTTPConnectToListenerServlet
Opens an HTTP connection to the Forms Listener servlet responsible for starting an instance of the Forms run time process.

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

ofsIconifyWindow
Adds the Window_Iconified property (with Enabled attribute) to the current message.

ofsIndexKey
Adds the Index_Key property to the current message.

ofsIndexSKey
Adds the Index_SKey property to the current message.

ofsInitSessionCmdLine
Adds the INITIAL CMDLINE property to the current message.

ofsInitSessionTimeZone
Adds the Time_Zone property to the current message.

ofsListItemValue
Adds the List_Item property of a PopList or a TList control to the current message.

ofsLoadValue
Loads the values of a byte array or a string array associated with a GUI control.

ofsLOVRequestRow
Adds the LOV_REQUEST_ROW property to the current message.

ofsLOVSelection
Adds the LOV_SELECTION property to the current message.

ofsMenuParamDlgOK
Adds the MENUPARAM_DLGOK property to the current message. This statement defines the text in the menu param dialog control.

ofsOpenWindow
Adds the Window_Open property (with Disabled attribute) to the current message.

ofsRemoveFocus
Adds the Focus property (with Disabled attribute) to the current message.

ofsSetCursorPosition
Adds the Cursor_Position property of a Text Field control to the current message.

ofsSetErrorDialogTitle
Adds the DISPLAYERRORDIALOG_TITLE property to the current message.

ofsSetFontName
Adds the Font_Name property to the current message.

ofsScroll
Adds the Block_Scroller property to the current message.

ofsScrollSize
Adds the Block_Scroller_Size property to the current message.

ofsSelectItem
Adds the Value property (with Enabled attribute) to the current Message.

ofsSelectMenuItem
Adds the Menu_Event property to the current message.

ofsSelectTreeEvent
Adds the Selected_Event property of a Tree Control to the current message.

ofsSendRecv
Sends the client request as Forms messages to the Forms server, gets the server response, and reads the responses as Forms messages.

ofsServerSideDisconnect
Disconnects QALoad's socket connection to the server-side code. The server-side code intercepts the messages between QALoad and the Forms Listener servlet.

ofsSetColorDepth
Adds the Color_Depth property to the current message.

ofsSetDisplaySize
Adds the Display_Size property to the current message.

ofsSetExpectedServerMsg
Enables the script to continue if a known error or warning message is received from the server.

ofsSetFontName
Adds the Font_Name property to the current message.

ofsSetFontSize
Adds the Font_Size property to the current message.

ofsSetFontStyle
Adds the Font_Style property to the current message.

ofsSetFontWeight
Adds the Font_Weight property to the current message.

ofsSetICXTicket
Sets the value of the ICX ticket for the current Oracle Applications login. The statement is used only in a Universal OFS-WWW session, as a replacement for the OracleAppsLogin() statement.

ofsSetInitialVersion
Adds the Initial_Version property to the current message.

ofsSetJavaContainerArgName
Adds the JAVACONTAINER_ARG_NAME property to the current message.

ofsSetJavaContainerArgValue
Adds the JAVACONTAINER_ARG_VALUE property to the current message.

ofsSetJavaContainerEvent
Adds the JAVACONTAINER_ARG_EVENT property to the current message.

ofsSetLogonDatabase
Adds the LOGON_DATABASE property to the current message.

ofsSetLogonPassWord
Adds the LOGON_PASSWORD property to the current message.

ofsSetLogonUserName

Adds the LOGON_USERNAME property to the current message.

ofsSetNoRequiredVAList
Adds the Required_VA_List property (with Disabled attribute) to the current message.

ofsSetPropertyBoolean
Adds the generic boolean property (with Enabled attribute) to the current message.

ofsSetPropertyByte
Adds the generic byte property to the current message.

ofsSetPropertyByteArray
Adds the generic byte array property to the current message.

ofsSetPropertyCharacter
Adds the generic Character property to the current message.

ofsSetPropertyDate
Adds the generic Date property to the current message.

ofsSetPropertyFloat
Adds the generic Float property to the current message.

ofsSetPropertyInteger
Adds the generic Integer property to the current message.

ofsSetPropertyPoint
Adds the generic Point property to the current message.

ofsSetPropertyRectangle
Adds the generic Rectangle property to the current message.

ofsSetPropertyString
Adds the generic String property to the current message.

ofsSetPropertyStringArray
Adds the generic String array property to the current message.

ofsSetPropertyVoid
Adds the generic Void property to the current message.

ofsSetRequiredVAList
Adds the Required_VA_List property (with Enabled attribute) to the current message.

ofsSetRunOptions
Sets the runtime values for CONNECT TYPE, HEARTBEAT, LOGGING (to replay capture file) and CHECK SERVER MESSAGES.

ofsSetScaleInfo
Adds the Scale property to the current message.

ofsSetScreenResolution
Adds the Screen Resolution property to the current message.

ofsSetSelection

Adds the Selection property of a Text Field control to the current message.

ofsSetServletMode
Creates a socket connection to the server-side code which communicates with the Forms Listener Servlet.

ofsSetServerFailedMsg
Enables QALoad to fail playback based on the user-entered string and filter parameters.

ofsSetValue
Adds a generic Value property to the current message.

ofsSetWindowLocation
Adds the Location property of a Window control to the current message.

ofsSetWindowSize
Adds the Size property of a Window control to the current message.

ofsShowWindow
Adds the Visible property (with Enabled attribute) to the current message.

ofsSocketDisconnect
Closes the connection of a socket-mode playback.

ofsStartSubMessage
Adds a sub-message to the current message.

ofsTabControlTopPage
Adds the TabControl_Top_Page property to the current message.

ofsUnSetPropertyBoolean
Adds a generic Boolean property (with Disabled attribute) to the current message.

OracleAppsLogin
This method simulates an Oracle Applications 11i login and retrieves the icx_ticket associated with that login. It should be performed once per virtual user.

PopupHelp
This method constructs a new OracleFormsMsg for this type of control.

ofsActivateListItem

Adds the TList_Activated property to the current message.

Tlist_Activated property indicates user selection of an item in a List control.

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.

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

```
ofsGetServerData();
```

Return Value

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. RR__printf is executed prior to DO_ExtractString to retrieve the filter parameters for DO_ExtractString.

```
.....
/* Declare Variables */
   char *Value = 0;
...
BEGIN_TRANSACTION();
...
...
   ofsSendRecv(1); //ClientSeqNo=1|CapTime=1090942125.437|1090942125.437


   RR__printf("reply data: %s",ofsGetServerData() );
```

```
   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*/
                     &Value      /* Value contains the dynamic value to be used in
subsequent statements */
                   );

RR__printf("item:  %s", Value);
```

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();
```

ofsHTTPSDoSSLHandshake

Establishes an SSL socket connection and starts an SSL handshake.

Syntax

```
void ofsHTTPSDoSSLHandshake();
```

Return Value

Parameters

None

Example

```
ofsHTTPSDoSSLHandshake();
```

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.
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.

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 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

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>sClassName</td><td>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.</td></tr>
<tr><td>iHandlerID</td><td>Captured ID of the GUI control for the current message.</td></tr>
<tr><td>iAction</td><td>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:
<table><tr><th>Value</th><th>Description</th></tr><tr><td>OFS_ADD</td><td>Add the property to the current message.</td></tr><tr><td>OFS_ENDMSG</td><td>Add the property to the current message and end the current message.</td></tr></table></td></tr>
<tr><td>iPropertyID</td><td>Oracle-designated ID for the property being added.</td></tr>
<tr><td>sFormsVersion</td><td>OFSFormsVersionEnum
The Forms Version of the captured application.. Valid values are:
<table><tr><th>Value</th><th>Description</th></tr><tr><td>FORMS_10g_SERVLET</td><td>Oracle Application Server 10g</td></tr><tr><td>FORMS_9i_SERVLET</td><td>Oracle Application Server 9i</td></tr><tr><td>FORMS_6i_SERVLET</td><td>Oracle Forms 6i Servlet Mode</td></tr><tr><td>FORMS_6i_11i_SERVLET</td><td>Oracle Forms 6i in 11i Environment</td></tr><tr><td>FORMS_6i_SOCKET</td><td>Oracle Forms 6i Socket Mode</td></tr><tr><td>FORMS_60_SOCKET</td><td>Oracle Forms 6.0</td></tr></table></td></tr>
</table>

Example

```
ofsSetInitialVersion( "RUNFORM", 1, OFS_ADD, 268, "60818" );
```

ofsSetJavaContainerArgName

Adds the JAVACONTAINER_ARG_NAME property to the current message.

This statement defines the name assigned to an item in a JavaContainer control.

Syntax

```
void ofsSetJavaContainerArgName(const char *sHandlerName, int iControlId, int iAction, int iPropertyID, const char *sValue);
```

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:
Value — Description
OFS_SOCKET — Socket connection
OFS_HTTP — HTTP connection
OFS_HTTPS — Secure (SSL) connection</td></tr>
<tr><td>iHearbeatInterval</td><td>Heartbeat interval (seconds).</td></tr>
<tr><td>iCheckServerMsgs</td><td>OFSMessageCheckingEnum</td></tr>
</table>

	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);
```

OracleAppsLogin

Simulates an Oracle Applications 11i login (Personal Home Page) and retrieves the icx_ticket associated with that login.

This statement appears on the script when the Convert option for OracleAppsLogin is enabled. The URL parameter should be the actual Oracle Personal Home Page address. Exceptions are thrown if the page cannot be found or if the icx_ticket isn't returned by the server.

Syntax

```
OracleAppsLogin(const char *ServerURL, const char *UserID, const char *Password)
```

Parameters

Parameter	Description
ServerURL	URL of the Oracle Apps Login page (the Personal Home Page address).
UserID	User ID that was entered on the login page during the recording.
Password	Password that was entered on the login page during the recording.

Example

```
//QALoad posts the login information to the homepage URL, retrieves the
//ICX ticket from the reply and uses the ICX ticket in a subsequent Forms message.
OracleAppsLogin( http://app11i..com:8000/oraclemypage.home, "myuserid", "mypassword" );
```

PopupHelp

OracleFormsMsg

Constructs a new OracleFormsMsg for this type of control.

Syntax

```
PopupHelp( handlerID );
PopupHelp( controlName );
```

Return Value

An OracleFormsMsg.

Parameters

Parameter	Description
HandlerId int	Control number. Each control on a form is given a unique number by the server. The *QA*Load Script Development Workbench automatically creates constants for this value that refer to the specific control by name and/or type.
ControlName string	Control name assigned by the original developer of the Oracle Forms application. To use control names, the original script must be recorded with the "record=names" option.

Exceptions

```
QALoad Exception ("No match for control name: <control_name>'");
```

Example

```
oracleFormsMsg1 = oracleForms.PopupHelp( CONTROL_001 );
oracleFormsMsg1 = oracleForms.PopupHelp( "CONTROLabc" );
```

QALoad

QALoad 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 conjuction 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 deallocated or uninitialized. When transaction restarting occurs for a failed transaction, QALoad will first execute any code starting after the call to DO_SetTransactionCleanup allowing you to clean up important information and prevent memory leaks before retrying the transaction.

DO_SetTransactionStart
Defines a point at the beginning of the transaction loop that QALoad uses to rewind the transaction if the transaction fails and Restart Transaction error handling is selected in the QALoad Conductor.

DO_SetValue
Associates a value to a variable name. Variable names are embedded into parameter strings of QALoad functions and the value is interpolated at replay. Currently, DO_Http and DO_Https are the only functions that interpolate the variables.

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

END_TRANSACTION
This command marks the end of the transaction loop.

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

EXIT
Stops script processing and returns control back to the Conductor.

GET_ABSOLUTE_VUNUM
Gets the absolute virtual user number.

GET_DATA
Requests that QALoad Conductor send the next datapool record to the script.

GET_DATA_FIELD
Accesses the fields from the data record that was just read using the READ_DATA_RECORD statement. Field numbering starts at 1.

GET_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.

RANDOM_NUMBER
Returns a string representation of a pseudo-random random number.

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

READ_DATA_RECORD
Reads a data record from a local datapool file.

RND_DELAY
Delays the script for a random interval before proceeding.

RND_DELAY_RANGE
Delays the script for a random interval, within a specified range, before proceeding.

RR__FailedMsg
Outputs a fatal error message to the Conductor.

RR__GetDebugFlag
Gets the debug flag for the script.

RR__printf

Prints formatted output to the standard output stream.

SET_ABORT_FUNCTION
Registers a callback function within the virtual user to call whenever the test operator manually aborts a test from the QALoad Conductor.

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 128 characters.

Example

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

CLOSE_ALL_DATA_POOLS

Closes all open local datapool files.

All local datapool files should be closed at the end of the script using this statement.

Syntax

```
CLOSE_ALL_DATA_POOLS ();
```

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

0 or greater 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.

Syntax

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

Return Value

TRUE if successful

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 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
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 = 0;

...

DO_Http("GET http://www.host.com/ HTTP/1.1\r\n\r\n");
/*
 * The page returns a page containing "<title>Enter Login</title>"
 */

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

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

/*
 * prints "The extracted title: Enter Login"
 */
```

DO_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 128 characters.

Example

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

EXIT

Stops script processing and returns control back to the Conductor.

Syntax

```
EXIT ( );
```

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

oad

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* frame, int pNumber, int rewindflag);
```

Return Value

Parameters

Parameter	Description
frame	The name of the datapool file, including a drive and path.
pNumber	The ID that was given to the datapool when it was inserted into the script. This is used anytime the script reads a record or field from the datapool.
rewindFlag	TRUE if the datapool file should be rewound to the beginning after it reaches the end. FALSE if it should not rewind.

Example

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

/* Default placement before BEGIN_TRANSACTION */
SYNCHRONIZE();
BEGIN_TRANSACTION();
```

RANDOM_NUMBER

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

The seed value that is used to generate the pseudo-random random number is set within the Conductor.

Syntax

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

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 pseudo-random random set of alpha or alphanumeric characters of the specified width.

The seed value that is used to generate the pseudo-random random number is set within the Conductor.

Syntax

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

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
comp
Unicode Uni

```
Real numbers:
                    251.736600
                    251.74 2.517366e+002
                    2.517366E+002
Address as:
                    0012FFAC
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 4.x

SAP 4.x Commands

Do_SAPCheckScreen
Used as a synchronization point in your QALoad script.

Do_SAPCheckStatus
Do_SAPCheckStatus compares the status message at the bottom of the SAP window with an expected message fragment.

Do_SAPCheckTitle
The Do_SAPCheckTitle function compares the SAP screen title with some expected text.

Do_SAPClearMessages
Do_ SAPClearMessages checks the title of the SAP window and the SAP status message at the bottom.

Do_SAPDumpEvent
Prints the current values of the IT_EVENT structure to the Player window.

Do_SAPExtractString
Allows you to specify a string to search (src), a filter string, and a left and right filter marker, and will return the number of characters copied into the destination variable (dest).

Do_SAPFrontEnd
Enables graphical display for replay of scripts.

Do_SAPFullMenu
Enables full or dynamic menus during SAP logins.

Do_SAPGetControlValue
Used to retrieve the value for the specified control.

Do_SAPGetIt_Event
Retrieves a pointer to the IT_EVENT structure (used in GUILIB.H).

Do_SAPGetScreenTitle
Retrieves a pointer to the current screen title from the current IT_EVENT structure.

Do_SAPGetStatusMsg
Retrieves a pointer to the status message from the current IT_EVENT structure.

Do_SAPInit

This function is required, and initializes the thread local storage for each SAP thread.

Do_SAPLogging
Turns the detail logging capability on and off.

Do_SAPLogin
Connects and logs into the SAP application server. If the script is unable to connect to the server, an error message is displayed and the script is aborted.

Do_SAPLogoff
Logs the script off of the application server.

Do_SAPSendDblClick
Whenever the user double-clicks when using the SAP GUI during capture, a Do_SAPSendDblClick event is recorded to the capture file.

Do_SAPSendEvent
Causes the script to send the current event to server.

Do_SAPSendMenu
Locates the Menu name provided in the Menu string and sets the menu option in the current event structure. The menu string for multi-level menus is formed by appending a -> string to each menu level, such as Edit->Copy, or File->open. It then sends the event to the server.

Do_SAPSendOKCode
Sets the OK code field in the IT_EVENT structure. The event is then automatically sent to the server.

Do_SAPSendPFKey
Duplicates the sending of a PFKey by the application by setting the PFKey value in the IT_EVENT structure and then sending the transaction to the server.

Do_SAPSendReturn
Sets the VK_RETURN PFKey in the event structure and sends the event to the server. This is the same as pressing Return or as the Green Check mark on the toolbar.

Do_SAPSetCheckScreenWildcard
Specifies the wildcard character to use for wildcard matching with Do_SAPCheckScreen.

Do_SAPSetCtrlValue
Sets the value for a control.

Do_SAPSetCursor
Sets the cursor fields in the IT_EVENT structure to the specified control.

Do_SAPCheckScreen

Used as a synchronization point in your QALoad script. Should be called after each Do_SAPSendEvent() to ensure that the screen returned by the server is the one expected by the script.

Each event sent from the SAP application server to the client includes the name of the ABAP program and the current screen title. This command ensures that the script and the application do not get out of sync.

Use Do_SAPCheckScreen with Do_SAPSetCheckScreenWildcard to perform a wildcard search for a screen title. This is especially useful if the screen title is likely to change with each new entry/lookup in the database during replay. If you insert the wildcard (set in Do_SAPSetCheckScreenWildcard) as the first character of the title, then all titles with the same right-most characters will match. If the wildcard is located in any character position other than the first, QALoad does not treat it as a wildcard. For example, "*est" will match "test," "tempest," and "est," but will not match "tester." This prevents possible conflicts when a wildcard character is present in a captured string, but is not intended to be a wildcard. This also prevents conflicts within pre-existing scripts that were converted before the wildcard matching option was added in Release 4.4.

If the end of a title is problematic during replay, it isn't necessary to use a wildcard match. Instead, reduce the number of characters that are compared in the title. For example, if the order number in the title "ORDER# PROCESSED: 12345" is likely to change during replay, shorten the title to remove the characters that are changing. For example, shorten the title to "ORDER# PROCESSED:". This will result in a match with any title during replay that contains the first characters "ORDER# PROCESSED:".

Syntax

```
Do_SAPCheckScreen ( 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

```
Do_SAPCheckScreen( "MMR1", "SAPLMGMM", "Create Raw Material: Initial Screen" );
```

Do_SAPCheckStatus

Compares the status message at the bottom of the SAP window with an expected message fragment.

The second parameter of the function is the display flag; TRUE prints out the status message when it is unexpected.

Syntax

```
Do_SAPCheckStatus( char* szExpected );
```

Return Value

1 if the string appears anywhere in the message

0 if the string does not appear in the message.

Parameters

Parameter	Description
szExpected	A string containing the current string status.

Example

```
Do_SAPSendPFKey("Save"); // Click the Save button
CHECK_POINT( TIMER9, 33); // Record response time for saving document
if (!Do_SAPCheckStatus("has been saved")) abort_function();
// Abort due to an
//unexpected status
//message
```

Do_SAPCheckTitle

Compares the SAP screen title with some expected text.

Syntax

```
Do_SAPCheckTitle( char* szExpected );
```

Return Value

1 if the text appears anywhere in the SAP title

0 if the text does not appear anywhere in the SAP title

Parameters

Parameter	Description
szExpected	A string containing the current string title.

Example

```
Do_SAPSendReturn();
if (Do_SAPCheckTitle("PO Extension")) Do_SAPSendReturn();
// This screen may sometimes appear. Press Return if it does.
```

Do_SAPClearMessages

Checks the title of the SAP window and the SAP status message at the bottom.

If the window title is "Information", then an information popup box has appeared and a return is sent to clear it. If the status message includes "W:", then a warning has appeared. Press Return to clear the warning. Do_SAPClearMessages loops until no more information boxes or warnings appear.

Syntax

```
Do_SAPClearMessages();
```

Return Value

Parameters

None.

Example

```
Do_SAPSetCtrlValue ( "VBAK-AUART", 0, "yta " );
// Enter data into SAP field.

Do_SAPSendReturn();
// Press Return to validate input.

Do_SAPClearMessages();
// Clear any warnings and information boxes
// and display the messages.

Do_SAPSendPFKey("Save");
// Click the Save button.

CHECK_POINT( TIMER9, 1);
// Record response time for saving document.
```

Do_SAPDumpEvent

Prints the current values of the IT_EVENT structure to the Player window.

This command is used for debugging purposes, primarily to view control and function key names and values.

Syntax

```
Do_SAPDumpEvent();
```

Return Value

Parameters

None.

Example

```
Do_SAPDumpEvent();
```

Do_SAPExtractString

Allows you to specify a string to search (src), a filter string, and a left and right filter marker. Returns the number of characters copied into the destination variable (dest).

Syntax

```
int Do_SAPExtractString ( char* dest, char* src, char* filter, unsigned short leftmarker,
unsigned short rightmarker );
```

Return Value

Parameters

Parameter	Description
dest	Pointer to destination string.
src	Pointer to string to search.
filter	Pointer to filter string.
leftmarker	Left filter marker.
rightmarker	Right filter marker.

Example

```
char src[64];
char dest[64];
char filter[64];
strcpy ( src, "Inventory number = 123456" );
strcpy ( filter, "Inventory number = <>" );
Do_SAPExtractString ( dest, src, filter, '<', '>' );
printf ( "dest = [%s]\n", dest );
```

Output

```
"dest = [123456]"
```

Do_SAPFrontEnd

Enables graphical display for replay of scripts.

Syntax

```
Do_SAPFrontEnd( SAPEnableFrontEndEnum flag );
```

Return Value

Parameters

Parameter	Description
flag	*SAPEnableFrontEndEnum* Full menu flag. Valid values are: Value — Description TRUE — Enables SAP graphical display FALSE — Disables SAP graphical display

Example

```
Do_SAPFrontEnd( TRUE );
```

Do_SAPFullMenu

Enables full or dynamic menus during SAP logins.

Syntax

```
Do_SAPFullMenu( SAPFullMenuFlagEnum flag );
```

Return Value

Parameters

Parameter	Description	
`flag`	SAPFullMenuFlagEnum	
	Value	Description
	TRUE	Causes SAP logins with full menus (all the menu structure is downloaded from the application server at once)
	FALSE	Causes SAP logins with dynamic menus (menus are downloaded as the menus are used)

Example

```
Do_SAPFullMenu( FALSE );
```

Do_SAPGetControlValue

Used to retrieve the value for the specified control.

The control is specified by name, with the occurrence parameter distinguishing between multiple controls of the same name. The Do_SAPDumpEvent() command can be used to display a list of controls on the current form. If the control is a radio button or a check button, the literals TRUE or FALSE are placed in the buffer indicating if the control is currently selected.

Syntax

```
int Do_SAPGetControlValue( const char* CtrlName, int CtrlOcc, char* Buffer );
```

Return Value

TRUE (1) if the control is found.

FALSE (0) if the control is not found .

Parameters

Parameter	Description
`CtrlName`	`Contains the name of the control.`
`CtrlOcc`	`Shows the occurrence of the control (0=first occurrence).`

Buffer	A pointer to a buffer to receive the current contents of the control. Note that no length checking is performed, so ensure that the buffer is large enough to contain the returned value.

Example

```
char buff[1024];
Do_SAPGetControlValue ( "RMMG1-MATNR", 1, buff );
```

Do_SAPGetIt_Event

Retrieves a pointer to the IT_EVENT structure (used in GUILIB.H).

Syntax

```
PIT_EVENT Do_SAPGetIt_Event();
```

Return Value

Parameters

None.

Example

```
PIT_EVENT temp;
temp = Do_SAPGetIt_Event ();
printf ("\nScreen Title = %s\n", temp->szNormTitle);
```

Do_SAPGetScreenTitle

Retrieves a pointer to the current screen title from the current IT_EVENT structure.

Syntax

```
char *Do_SAPGetScreenTitle ();
```

Return Value

Parameters

None.

Example

```
char *Msg;
Msg = Do_SAPGetScreenTitle ( );
```

Do_SAPGetStatusMsg

Retrieves a pointer to the status message from the current IT_EVENT structure.

This command is typically used after a Do_SAPSendEvent() statement to check for possible returned error messages.

Syntax

```
char *Do_SAPGetStatusMsg();
```

Return Value

Parameters

None.

Example

```
char *Msg;
Msg = Do_SAPGetStatusMsg();
```

Do_SAPInit

This function is required, and initializes the thread local storage for each SAP thread.

Syntax

```
Do_SAPInit( PLAYER_INFO* s_info );
```

Return Value

Parameters

Parameter	Description
s_info	Structure used by each virtual user.

Example

```
Do_SAPInit( s_info );
```

Do_SAPLogging

Turns the detail logging capability on and off.

Logging is a debugging tool that logs all SAP events received from the server to a log file named saplg###.log, where ### is the virtual user number on the player. The log file is a binary file that can only be viewed using the QALSAP program.

The flag parameter is used to turn logging off and on or to clear simultaneously any previous log entries. Once logging is turned on, a log record is written before each event is sent to the SAP server and after each event is received from the server.

Syntax

```
Do_SAPLogging( SAPLoggingFlagsEnum flag );
```

Return Value

Parameters

Parameter	Description	
flag	SAPLoggingFlagsEnum	
	Value	Description
	SAPLOG_ON	Turn SAP logging on
	SAPLOG_OFF	Turn SAP logging off
	SAPLOG_CLEAR	Clear the SAP log

Example

```
Do_SAPLogging( SAPLOG_ON );
```

Do_SAPLogin

Connects and logs into the SAP application server. If the script is unable to connect to the server, an error message is displayed and the script is aborted.

Syntax

```
Do_SAPLogin( const char* hostname, const char* systemNo, const char* clientNo, const char*
Name, const char* Pwd, const char* Lang);
```

Return Value

Parameters

Parameter	Description
hostname	The Host name of the SAP application server.
systemNo	The system number to login to.
clientNo	The client number to use (may be defaulted to "").
Name	The user ID.
Pwd	The password.
Lang	The language (may be defaulted to "").

Example

```
Do_SAPLogin( "/H/204.79.199.5/H/207.213.200.19", "01", "850", "compuware", "qaload", "" );
```

Do_SAPLogoff

Logs the script off of the application server.

Syntax

```
Do_SAPLogoff();
```

Return Value

Parameters

None.

Example

```
Do_SAPLogoff();
```

Do_SAPSendDblClick

Whenever the user double-clicks when using the SAP GUI during capture, a Do_SAPSendDblClick event is recorded to the capture file.

Syntax

```
Do_SAPSendDblClick ();
```

Return Value

Parameters

None.

Example

```
Do_SAPSetCursor( "RMMG1-MBRSH", 0 );  /* Value: Industry s */
Do_SAPSendDblClick();
```

Do_SAPSendEvent

Causes the script to send the current event to the server.

Typically, the script sets the value of one or more controls and indicates the action to be taken by setting a function key using Do_SAPPfKey(). The Do_SAPSendEvent function completes the screen by transmitting

the updated information and action code to the server. Upon completion of the Do_SAPSendEvent() function, the script automatically retrieves the server's reply.

Syntax

```
Do_SAPSendEvent();
```

Return Value

Parameters

None.

Example

```
Do_SAPSendEvent();
```

Do_SAPSendMenu

Locates the Menu name provided in the Menu string and sets the menu option in the current event structure.

The menu string for multi-level menus is formed by appending a "->" string to each menu level, such as Edit->Copy, or File->open. It then sends the event to the server.

Syntax

```
Do_SAPSendMenu( const char* MenuString );
```

Return Value

Parameters

Parameter	Description
MenuString	The (case insensitive) name of the menu option to set.

Example

```
Do_SAPSendMenu( "Log off" );
```

Do_SAPSendOKCode

Sets the OK code field in the IT_EVENT structure. The event is then automatically sent to the server.

Syntax

```
Do_SAPSendOKCode ( const char* OKcode );
```

Return Value

Parameters

Parameter	Description
OKCode	A string containing the new OK code to send.

Example

```
Do_SAPSendOKCode( "/nmmr1" );
```

Do_SAPSendPFKey

Duplicates the application sending a PFKey by setting the PFKey value in the IT_EVENT structure, and then sending the transaction to the server.

The key name supplied as a parameter to Do_SAPSendPFKey() must be present in the current screen's IT_EVENT structure. If it isn't, an error message is given and the script is aborted.

To send a value that isn't one of the currently available key names, call this function with the decimal virtual key value (iVKValue) preceded by a #. For example, to send an F1 whether or not it was defined in the IT_EVENT structure, call Do_SAPSendPFKEY("#112").

Syntax

```
Do_SAPSendPfKey ( const char* KeyName );
```

Return Value

Parameters

Parameter	Description
KeyName	The name of the key to set (case insensitive).

Example

```
Do_SAPSendPFKey ( "Possible entries" );
```

Do_SAPSendReturn

Sets the VK_RETURN PFKey in the event structure and sends the event to the server. This is the same as pressing Return or as the Green Check mark on the toolbar.

Syntax

```
Do_SAPSendReturn();
```

Return Value

Parameters

None.

Example

```
Do_SAPSendReturn();
```

Do_SAPSetCheckScreenWildcard

Specifies the wildcard character to use for wildcard matching with Do_SAPCheckScreen.

Although all converted scripts include Do_SAPSetCheckScreenWildcard ('*') as one of the first functions, you can specify a different wildcard character to use later in the script.

Syntax

```
Do_SAPSetCheckScreenWildcard ( unsigned short wildcard );
```

Return Value

Parameters

Parameter	Description
wildcard	A character to match in Do_SAPCheckScreen.

Example

```
Do_SAPSetCheckScreenWildcard ('*') ;
```

Do_SAPSetCtrlValue

Sets the value for a control.

If the control is a radio-button, then all other radio-buttons in that control's group are cleared.

Syntax

```
Do_SAPSetCtrlValue ( const char* CtrlName, int CtrlOcc, char* Buffer );
```

Return Value

Parameters

Parameter	Description
CtrlName	The name of the control.
CtrlOcc	Occurrence of the control (0=first occurrence).
Buffer	Pointer to a buffer containing the value to load into the control. If the control is a radio button or a check button, then the buffer must contain the string TRUE to set the button, or FALSE to clear the button.

Example

```
Do_SAPSetCtrlValue ( "RMMG1-MATNR", 1, "B123458" );
```

Do_SAPSetCursor

Sets the cursor fields in the IT_EVENT structure to the specified control.

If the control appears multiple times on the screen, use the ctrlOcc field to specify which control the cursor is to be set to. CtrlOcc is zero-based, that is, the first occurrence is 0, then the second is 1, and so forth. To set the cursor to an arbitrary point on the screen, the ctrlName parameter may be entered in a numeric format "#row,col (i.e. "#2,1").

Syntax

```
Do_SAPSetCursor( const char* ctrlName, int ctrlOcc );
```

Return Value

Parameters

Parameter	Description
ctrlName	The name of the control to set the cursor to.
ctrlOcc	The occurrence of the control name.

Example

```
Do_SAPSetCursor( "MSICHTAUSW-KZSEL ", 0 ); /* Value: Variant */
```

SAP 6.x

SAP 6.x 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.

SAPGuiCreateColl
Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. This call creates a collection.

SAPGuiDestroyColl
Specifies the method (or property) that is called (or set) in the object specified in the previous SAPGuiPropIdStr call. This call destroys a collection and decrements the reference count.

SAPGuiPropIdStr
Specifies the object ID string to use with subsequent SAPGui calls.

SAPGuiPropIdStrExists
Specifies the object ID string to use with subsequent SAPGui calls.

SAPGuiPropIdStrExistsEnd
Marks the end of the block of code that is executed if the condition in a prior SAPGuiPropIdStrExists command is true.

SAPGuiSessionInfo
Specifies the method (or property) that will be called, allowing the script access to the SAP GuiSessionInfo objects.

SAPGuiSetCheckScreenWildcard
Specifies the wildcard character to use for wildcard matching with `SAPGuiCheckScreen`.

SAPGuiVerCheckStr
Specifies the SAP GUI frontend version number at the time the capture file was made.

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

x

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");
```

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" );
```

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

Uniface

Uniface Commands

BEGIN_UENTITY
Begins the declaration of the UNIFACE entity.

DO_Logfile_URB
Controls whether or not to generate a log file showing load test information, such as requests and responses.

DO_URB_AsciiToHex
Converts the ASCII hexadecimal represented buffer into its binary representation.

DO_URB_Init
Sets all necessary internal variables needed to load test a UNIFACE script.

DO_URB_setoprretry
Sets the number of times to retry an operation activation.

DO_URB_ubin2uf
Converts binary data to a UNIFACE format.

DO_URB_udbl2uf
Converts a double float to a UNIFACE format.

DO_URB_uecreate
Creates a UNIFACE environment.

DO_URB_uedelete
Closes a UNIFACE environment.

DO_URB_uentcreo
Creates an occurrence and makes it current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_uentoccs
Returns the number of occurrences that exist in an entity.

DO_URB_uentseto
Makes an occurrence current. Note that this function can only be used with entity parameters and not with occurrence parameters.

DO_URB_ufreeh
Deletes a handle.

DO_URB_uinstdel
Deletes a component instance.

DO_URB_uinstnew
Creates an instance of a component.

DO_URB_uinstopr
Returns a handle to an operation.

DO_URB_ulist2uf
Converts an item list to a UNIFACE format.

DO_URB_ulistdel
Deletes an item.

DO_URB_ulistfree
Frees a UNIFACE list.

DO_URB_ulistget
Gets an item.

DO_URB_ulistnew
Creates an item list.

DO_URB_ulistput
Puts an item.

DO_URB_ulistputlist
Copies an item from a specified source to the items of a list.

DO_URB_ulistputx
Puts an item.

DO_URB_ulong2uf
Converts a long to a UNIFACE format.

DO_URB_unifree
Frees memory.

DO_URB_uniname
Returns the name. Caller supplies allocated memory for name. Field names are not supported.

DO_URB_uopract
Activates an operation.

DO_URB_uoprprms
Returns the number of parameters.

DO_URB_uprmdir
Returns the direction of a parameter.

DO_URB_uprmgeth
Gets a reference to a parameter of an operation or a field of an entity. Or detaches a parameter from an operation.

DO_URB_uprmtype
Returns the data type of a parameter or entity field.

DO_URB_ustr2uf
Converts a string to a UNIFACE format.

DO_URB_uuf2bin
Converts a UNIFACE format to binary data.

DO_URB_uuf2dbl
Converts a UNIFACE format to a double float.

DO_URB_uuf2list
Converts a UNIFACE format to item list.

DO_URB_uuf2long
Converts a UNIFACE format to a long.

DO_URB_uuf2str
Converts a UNIFACE format to string.

END_UENTITY
Ends the declaration of a UNIFACE entity.

UFIELD
Declares or defines a UNIFACE field.

BEGIN_UENTITY

Begins the declaration of a Uniface entity.

Syntax

```
BEGIN_UENTITY(UNIFACE_ENTITY* entname, char* name, char* model, char* type, char* trxdef);
```

Return Value

Parameters

Parameter	Description
entname	The pointer to the UNIFACE_ENTITY structure.
name	The name of the Uniface entity.
model	The name of the Uniface model.
type	The type of the Uniface entity.
trxdef	The definitions of the transactions that are defined for the entity.

Example

```
BEGIN_UENTITY(race_formula1, "RACE", "FORMULA1", "SZ1",
"RACE,FORMULA1,SZ1,RACE_ID,N2,M,10,100,4,1.101,U_VERSION,S2,0,100,1,TRACK_CD,S2,0"
",100,3,RACE_DATE,D2,0,102,8,RACE_NAME,S2,0,100,40,RACE_DISTANCE,N2,10,100,2,RACE"
"_CONDITIONS,S2,0,100,80,RACE_NOTES,S2,128,100,0,0,0,0,0,,0,0,0,,0,0,0,,")
```

DO_Logfile_URB

Controls whether or not to generate a log file showing load test information, such as requests and responses.

If this command is specified, it generates one log file for every virtual user running the script. Log files appear in the script directory in the form URBnnnn.cap, where nnnn indicates the zero-based number of the virtual user executing this script. By default, this option is disabled. This command is automatically included in the script by QALoad 's Convert facility.

Syntax

```
int DO_Logfile_URB(int flag);
```

Return Value

0 (zero)

Parameters

Parameter	Description
flag	Turns logging on or off (TRUE/FALSE).

Example

```
DO_Logfile_URB(TRUE);
```

DO_URB_AsciiToHex

Converts the ASCII hexadecimal represented buffer into its binary representation.

Syntax

```
char * DO_URB_AsciiToHex(char * data);
```

Return Value

Pointer to the binary representation of the string.

Parameters

Parameter	Description
data	Pointer to a null terminated string containing the hexadecimal values of the bytes to be converted to binary.

Example

```
strcpy( urb_buffer,"0212035903107248");
DO_URB_AsciiToHex( urb_buffer );
DO_URB_ubin2uf( 4, 8, urb_buffer, 8 );
```

DO_URB_Init

Sets all necessary internal variables needed to load test a Uniface script.

Syntax

```
long DO_URB_Init(PLAYERINFO *s_info);
```

Return Value

0 (zero)

Parameters

Parameter	Description
s_info	Pointer to a PLAYERINFO structure

Example

```
DO_URB_Init( s_info );
```

DO_URB_setoprretry

Sets the number of times to retry an operation activation.

Syntax

```
long DO_URB_setoprretry(int Retries, int Sleep);
```

Return Value

0 (zero)

Parameters

Parameter	Description
Retries	Number of times to retry operation.
Sleep	Time to sleep between retries.

DO_URB_ubin2uf

Converts binary data to a Uniface format.

Syntax

```
long DO_URB_ubin2uf(int nHandle,int seqNr,char extData,long nLen);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description

nHandle	Integer handle to an operation or entity.
seqNr	Integer sequence number.
extData	Char external binary data.
nLen	Long external data length.

Example

```
strcpy( urb_buffer,"0212035903107248");
DO_URB_AsciiToHex( urb_buffer );
DO_URB_ubin2uf( 4, 8, urb_buffer, 8 );
```

DO_URB_udbl2uf

Converts a double float to a Uniface format.

Syntax

```
DO_URB_udbl2uf(nHandle,seqNr,dData);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Integer handle to an operation or entity.
seqNr	Integer sequence number.
dData	External double float data.

Example

```
DO_URB_udbl2uf(3,1,12345);
```

DO_URB_uecreate

Creates a Uniface environment.

This function sets up a Uniface environment based on the configuration parameters: command line, assignment file, and working directory. Only one Uniface environment is allowed per process. The application start-up shell parameter apsName is ignored.

Syntax

```
DO_URB_uecreate(runmode,hInstance,cmdLine,asnName,apsName, workDir,envHandle)
```

Return Value

1 = Success

-1 = Load Error

-2 = License Error

-4 = Allocation Error

Parameters

Parameter	Description
runMode	How Uniface is executed (batch or interactive). Always set this to 1.
hInstance	Instance handle. hInstance can be set to 0.
cmdLine	Command line.
asnName	.asn file name.
apsName	.aps file name.
workDir	Working directory.
envHandle	Uniface environment handle.

Example

```
DO_URB_uecreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formula1\\formula1.asn", "", "c:\\u@training\formula1\formula1", 0 );
```

DO_URB_uedelete

Closes a Uniface environment.

Syntax

```
long DO_URB_uedelete(int envHandle,int level);
```

Return Value

1 = Success

-1 = Load Error

-2 = License Error

-4 = Allocation Error

Parameters

Parameter	Description
envHandle	Uniface environment handle
level	Shutdown level.

Example

```
...
...
DO_URB_uecreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formula1\\formula1.asn", "", "c:\\u@training\formula1\formula1", 0 );
...
...
...
DO_URB_uedelete( 0, -1 );
...
...
```

DO_URB_uentcreo

Creates an occurrence and makes it current. Note that this function can only be used with entity parameters and not with occurrence parameters.

Syntax

```
long DO_URB_uentcreo(int nHandle,long occNr);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to the entity.
occNr	Sequence number of the occurrence.

Example

```
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
DO_URB_uinstopr( 1, "STORE ", 0, 2 ); |
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_uentcreo( 3, 1 ); / * creates first occurrence */
DO_URB_ustr2uf( 3, 1, "Field 1 data" );
DO_URB_ustr2uf( 3, 2, "Field 2 data" );
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
...
...
```

DO_URB_uentoccs

Gets the number of occurrences that exist in an entity.

Syntax

```
long DO_URB_uentoccs(int nHandle, long *occNr);
```

Return Value

0 = success

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to a Uniface entity.
occNr	(output) Number of occurrences that exist in the entity.

DO_URB_uentseto

Makes an occurrence current. Note that this function can only be used with entity parameters and not with occurrence parameters.

Syntax

```
long DO_URB_uentseto(int nHandle,long occNr);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an entity.
occNr	Sequence number of the occurrence.

Example

```
char *pString;
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_entseto( 3, 1 ); /* make occurrence 1 current*/
DO_URB_uuf2str( 3, 1, pString );
...
... /* do some manipulation with the returned string */
...
DO_URB_unifree( 3, pString ); /* free the memory */
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_ufreeh

Deletes a handle.

Syntax

```
long DO_URB_ufreeh(int nHandle);
```

Return Value

0 = success

<>0 = not successful

Parameters

Parameter	Description
nHandle	Any handle. Parameter hAny is always set to 0.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uinstdel

Deletes a component instance.

Syntax

```
long DO_URB_uinstdel(int nHandle);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to a component instance.

Example

```
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
...
...
DO_URB_uinstdel( 1 );
DO_URB_ufreeh( 1 )
```

DO_URB_uinstnew

Creates an instance of a component.

The parameter options is UDEFAULT_COMM_MODE, USYNC_COMM_MODE, and so on. The parameter propList is a NULL-character separated item list. The item list ends in two NULL characters. The parameters compID, and propList are optional.

Syntax

```
DO_URB_uinstnew(envHandle,compName,compID,instName,options, propList,newHandle);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
envHandle	Uniface environment handle.
compName	Component name.
compID	Component ID.
instName	Instance name.
options	Options.
propList	Property list.
newHandle	Instance handle.

Example

```
...
...
DO_URB_uecreate( 1, 0, "/ini=c:\\usys72\\bin\\usys.ini /pri=48",
"c:\\u@training\\formula1\\formula1.asn", "", "c:\\u@training\formula1\formula1", 0 );
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
```

DO_URB_uinstopr

Returns a handle to an operation.

Syntax

```
long DO_URB_uinstopr(int nHandle,long oprName,int newHandle);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to a component instance.
oprName	Operation name.
newHandle	Handle to the instance.

Example

```
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_ulist2uf

Converts an item list to a Uniface format.

Syntax

```
long DO_URB_ulist2uf(int nHandle,int SeqNr,int hList);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
SeqNr	Sequence number.
hList	Handle to item list.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_ulist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
```

DO_URB_ulistdel

Deletes an item.

This function corresponds with the Proc statement delitem. If UITEM_OPTION_NONE is specified, a value for index is expected and ID is ignored. If UITEM_OPTION_ID or UITEM_OPTION_ID_CASE is specified, index is ignored and a value for ID is expected.

Syntax

```
long DO_URB_ulistdel(int nHandle, int option, int index,char id);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an item list.
option	Option (UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE).
index	Item index.
id	pointer to an item ID.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uopract( 2 );
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
...
...
DO_URB_uuf2list( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ulistdel( 3, UITEM_OPTION_NONE, 1, "");
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistfree

Frees a Uniface list.

Syntax

```
long DO_URB_ulistfree(int Handle);
```

Return Value

0 = successful

< 0 = error

See the error descriptions that follow:

-201 Unknown protocol specified.
-202 An error was detected in the returned transaction. This error will be sent to the Player as a warning, since it may be normal.
-203 Invalid length of a parameter.
-204 A unexpected NULL parameter was passed to a call.
-205 Invalid entity. The entity passed as a parameter to a call is invalid. It may not have been opened using the DO_PSV_open call.
-206 Invalid field name.
-207 Invalid hit. The hit used in an update or in a fetch call is not the result of a DO_PSV_select call. The hit header is invalid.
-401 Invalid token found while parsing where clause or aggregate statements.
-501 No hit.

Parameters

Parameter	Description
`Handle`	`Handle to a list to be freed.`

DO_URB_ulistget

Gets an item.

This function corresponds with the Proc statement getitem. The parameter option is UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE. If UITEM_OPTION_NONE is specified, a value for index is expected and ID is ignored. If UITEM_OPTION_ID or UITEM_OPTION_ID_CASE is specified, index is ignored and a value for ID is expected.

Syntax

```
DO_URB_ulistget(int nHandle, UnifaceURBListOptionEnum option, int index, char* id, char** pItem)
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
`nHandle`	`Handle to an item list.`
`option`	UnifaceURBListOptionEnum List option. Valid values are: Value — Description UITEM_OPTION_NONE — Delete matching item index value. No item ID required

	UITEM_OPTION_ID	Delete item ID value. No item index required
	UITEM_OPTION_ID_CASE	Delete item ID by case value. No item index required
index	Item index.	
id	Pointer to an item ID.	
pItem	Item	

Example

```
char *pItem;
...
...
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uopract( 2 );
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
...
DO_URB_uuf2list( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ulistget( 3, UITEM_OPTION_NONE, 1, "", &pItem );
...
/* use pItem */
DO_URB_unifree( 3, pItem );
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistnew

Creates an item list.

Syntax

```
long DO_URB_ulistnew(int nHandle, int newHandle);
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
nHandle	Any handle.
newHandle	A handle to an item list.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
```

```
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_ulist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistput

Puts an item.

This function corresponds with the Proc statement putitem. The parameter option is UITEM_OPTION_NONE, UITEM_OPTION_ID or UITEM_OPTION_ID_CASE. If UITEM_OPTION_NONE is specified, a value for index is expected and ID is ignored. If UITEM_OPTION_ID or UITEM_OPTION_ID_CASE is specified, index is ignored and a value for ID is expected.

Syntax

```
long DO_URB_ulistput(int nHandle, UnifaceURBListOptionEnum option, int index, char*
id,char** item);
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
nHandle	Handle to an item list.
option	UnifaceURBListOptionEnum List option. Valid values are: Value — Description UITEM_OPTION_NONE — Delete matching item index value. No item ID required UITEM_OPTION_ID — Delete item ID value. No item index required UITEM_OPTION_ID_CASE — Delete item ID by case value. No item index required
index	Item index.
id	Pointer to an item ID.
item	Pointer to an item.

Example

```
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 1, "", "TRACK_NAME" );
DO_URB_ulistput( 3, UITEM_OPTION_NONE, 2, "", "TRACK_MAP" );
...
DO_URB_ulist2uf( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ufreeh( 3 );
...
...
```

DO_URB_ulistputlist

Copies an item from a specified source to the items of a list.

Syntax

```
long DO_URB_ulistputlist(int nHandleDst, int index, char *id, int nHandleSrc);
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
nHandleDst	Handle to the destination entity.
index	Item index.
id	Pointer to an item ID.
nHandleSrc	Handle to the source entity.

DO_URB_ulistputx

Puts an item.

Syntax

```
long DO_URB_ulistput(int nHandle, char *id, char *item, int sepcntr);
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
nHandle	Handle to an item list.
id	Pointer to an item ID.
item	Pointer to an item.
sepcntr	Number of separators for list and sublists.

Example

```
DO_URB_ulistnew( 2, 3 );   /* FIELDS */
DO_URB_ulistputx( 3, "ORDER", "52", 2 );
```

DO_URB_ulong2uf

Converts a long to a Uniface format.

Syntax

```
long DO_URB_ulong2uf(int nHandle, int seqNr, long lData);
```

Return Value

0 = successful

<>0 not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
lData	External long data.

Example

```
DO_URB_ulong2uf(3,1,1234);
```

DO_URB_unifree

Frees memory.

Syntax

```
long DO_URB_unifree(int nHandle, void *pvoid);
```

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Any handle.
pvoid	Pointer to start address of allocated memory.

Example

```
char *pString;
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 ); /* get an entity handle */
DO_URB_entseto( 3, 1 ); /* make occurrence 1 current*/
DO_URB_uuf2str( 3, 1, pString );
...
...
/* do some manipulation with the returned string */
...
DO_URB_unifree( 3, pString ); /* free the memory */
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_uniname

Returns the name. Caller supplies allocated memory for name. Field names are not supported.

When working with an operation handle, if seqNr is 0, the name of the operation itself is returned. If seqNr is a legal parameter sequence number, the name of the parameter is returned.

Syntax

```
long DO_URB_uniname(int nHandle, int seqNr, int maxLen, char name);
```

Return Value

Parameters

Parameter	Description
nHandle	Handle to a component instance, operation, or parameter.
seqNr	Sequence number.
maxLen	Size of name.
name	Name of the instance.

Example

```
char sName[128];
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB _uniname( 2, 1, 128, sName );
```

```
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uopract

Activates an operation.

Syntax

```
long DO_URB_uopract(int nHandle);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_ustr2uf( 2, 1, "19990101" );
DO_URB_ustr2uf( 2, 2, "19991231" );
DO_URB_uopract( 2 );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uoprprms

Returns the number of parameters.

Syntax

```
long DO_URB_uoprprms(int nHandle,long *pPrmCount);
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation
pPrmCount	Pointer to number of parameters

Example

```
int nParameterCount;
...
...
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uoprprms( 2, &nParameterCount );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uprmdir

Returns the direction of a parameter.

Syntax

```
long DO_URB_uprmdir(int nHandle, int seqNr,int *pDirection );
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation.
seqNr	Sequence number.
pDirection	Pointer to parameter direction. The parameter pDirection is UPARM_INPUT for IN direction, UPARM_OUTPUT for OUT direction, and (UPARM_INPUT | UPARM_OUTPUT) for INOUT direction. It is not relevant whether or not parameter data is attached to the operation parameter list.

Example

```
int nDirection;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmdir( 3, &nDirection);
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_uprmgeth

Gets a reference to a parameter of an operation or a field of an entity. Or detaches a parameter from an operation.

Syntax

```
long DO_URB_uprmgeth(int nHandle, int seqNr, int bDetach, int newHandle );
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity parameter.
seqNr	Sequence number.
bDetach	Detach data option (TRUE/FALSE).
newHandle	Handle to a basic parameter, entity parameter or entity field. If parameter bDetach is TRUE, the data of handle nHandle is detached from the constructed handle nHandle. It is not allowed to detach a field from an entity.

Example

```
int nType;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmtype( 3, &nType);
...
...
DO_URB_ufreeh( 3 );
DO_URB_ufreeh( 2 );
```

DO_URB_uprmtype

Returns the data type of a parameter or entity field.

Syntax

```
long DO_URB_uprmtype(int nHandle, int *pType );
```

Return Value

0 = successful

<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to a parameter or entity field.
pType	Pointer to data type. The parameter pType is UTYPE_STRING, UTYPE_BOOLEAN, and so on.

Example

```
int nType;
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_uprmgeth( 2, 1, FALSE, 3 );
DO_URB_uprmtype( 3, &nType);
...
DO_URB_ufreeh( 3 );
...
```

DO_URB_ustr2uf

Converts a string to a Uniface format.

Syntax

```
long DO_URB_ustr2uf(int nHandle,int seqNr,char *string)
```

Return Value

0 = successful
<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
string	Pointer to an external string data.

Example

```
DO_URB_uinstnew( 0, "S_RACE", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "LIST", 0, 2 );
DO_URB_ustr2uf( 2, 1, "19990101" );
```

DO_URB_uuf2bin

Converts a Uniface format to binary data.

Syntax

```
long DO_URB_uuf2bin(int nHandle, int seqNr, char pExData, long *pnLen);
```

Return Value

0 = successful
<>0 = not successful

Parameters

Parameter	Description

nHandle	Handle to an operation or entity.
seqNr	Sequence number.
pExData	External (output) binary data. The parameter pExtData is allocated on the heap. It has to be freed with DO_URB_unifree.
pnLen	Pointer to external data length.

Example

```
char *pBinaryData;
long nLen;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "", "", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2bin ( 2, 4, &pBinaryData, &nLen );
...
...
DO_URB_unifree( 2 , pBinaryData );
DO_URB_ufreeh( 2 );
```

DO_URB_uuf2dbl

Converts a Uniface format to a double float.

Syntax

```
long DO_URB_uuf2dbl(int nHandle,int seqNr,double *pdData);
```

Return Value

0 = succesful
<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
pdData	Pointer to an external double float data.

Example

```
double dNumber;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2dbl ( 2, 2, &dNumber );
...
...
DO_URB_ufreeh( 2 );
```

DO_URB_uuf2list

Converts a Uniface format to item list.

Syntax

```
long DO_URB_uuf2list(int nHandle, int seqNr, int hList);
```

Return Value

0 = successful
<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity
seqNr	Sequence number
hList	Create the parameter hList with DO_URB_ulistnew before calling this function. After using the item list, free it using DO_URB_ulistdel

Example

```
char *pItem;
...
...
DO_URB_uinstnew(0,"S_SERVICE","","",1,"",1);
DO_URB_uinstopr(1, "DO_SOMETHING", 0, 2);
...
...
DO_URB_uopract( 2 );
...
...
DO_URB_ulistnew( 2, 3 ); /* FIELDS */
...
DO_URB_uuf2list( 2, 3, 3 ); /* Field Name "FIELDS" */
DO_URB_ulistget( 3, UITEM_OPTION_NONE, 1, "", &pItem );
...
/* use pItem */
DO_URB_unifree( 3, pItem );
DO_URB_ufreeh( 3 );
```

DO_URB_uuf2long

Converts a Uniface format to a long.

Syntax

```
long DO_URB_uuf2long(int nHandle, int seqNr, long *plData);
```

Return Value

0 = successful
<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity.
seqNr	Sequence number.
plData	Pointer to external long data.

Example

```
long lNumber;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2long ( 2, 1, &lNumber );
...
...
DO_URB_ufreeh( 2 );
...
...
```

DO_URB_uuf2str

Converts a Uniface format to string.

Syntax

```
long DO_URB_uuf2str(int nHandle,int SeqNr,char **pExString);
```

Return Value

0 = successful
<>0 = not successful

Parameters

Parameter	Description
nHandle	Handle to an operation or entity
SeqNr	Sequence number
pExString	External string data. The parameter pExString is allocated on the heap. It has to be freed with DO_URB_unifree.

Example

```
char *pString;
...
...
DO_URB_uinstnew( 0, "S_ANY", "", "","", 1, "", 1);
```

```
DO_URB_uinstopr( 1, "RETRIEVE ", 0, 2 );
DO_URB_uopract( 2 );
DO_URB_uuf2str ( 2, 3, &pString );
..
..
DO_URB_unifree( 2 , pString );
DO_URB_ufreeh( 2 );
```

END_UENTITY

Ends the declaration of a Uniface entity.

Syntax

```
END_UENTITY(UNIFACE_ENTITY* pointer, UNIFACE_ENTITY* pointee);
```

Return Value

Parameters

Parameter	Description
pointer	New pointer for the entity.
pointee	The original pointer used in BEGIN_UENTITY as entname.

Example

```
END_UENTITY(RACE_FORMULA1, race_formula1);
```

UFIELD

Declares or defines a Uniface field.

Syntax

```
UFIELD (char* name, char* type, int size, char* index);
```

Return Value

Parameters

Parameter	Description
name	Field name.
type	Data and interface type.
size	Length of the field.
index	If the field is a key for the entity, this field holds the index.

Example

```
UFIELD( "RACE_ID", "N2", 4, "1.101" );
```

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 to return the low byte of.

Example

```
unsigned char lbyte;
...
byte = LoByte(GetLocalPort(S3));
```

Log

Records the character string passed into the log file.

Syntax

```
void Log(char *line_to_be_logged)
```

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_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_GetUniqueString
Applies to HTTP and SSL requests. Used to parse the most recent HTTP server reply to get the contents of a string that occurs between the left and right input strings.

DO_GetUniqueStringEx
Applies to HTTP and SSL requests. Used to parse a null-terminated input string (search) to get the contents of a string that occurs between the left and right input strings.

DO_Http
Applies to HTTP requests. Executes an HTTP request in the script.

DO_HttpCleanup
Applies to HTTP and SSL requests. Performs all necessary cleanup operations when a script exits or the user terminates the script.

DO_HttpVersion
Applies to HTTP and SSL requests. Specifies the version to use in the requests sent during playback.

DO_InitHttp
Applies to HTTP and SSL requests. Also applies to Visual Scripting. Sets all necessary internal variables needed to load test an HTTP script.

DO_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_SaveReplyType
Applies to HTTP and SSL requests. Specifies types of replies to save.

DO_SetAssumedContentType
Applies to HTTP and SSL requests. Sets the default content type if the web server doesn't send a content type header.

DO_SetBaudRate
Returns the baud rate the virtual user will use.

DO_SetBaudRateEx
Returns the transmission rate the virtual user will use.

DO_SetCheckpointName
Sets the name of the next automatic checkpoint for the next DO_Http or DO_Https statement in the script.

DO_SetCookie
Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

DO_SetCookieEx
Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

DO_SetJavascriptCleanupThreshold
Applies to HTTP and SSL requests. Periodically QALoad will destroy its internal JavaScript model and recreate it. DO_SetJavascriptCleanupThreshold sets a count of the number of times JavaScript parsing is done before destroying and recreating the model.

DO_SetMaxBrowserThreads
Applies to HTTP and SSL requests. Specifies the number of concurrent connections to make for playback.

DO_SetMaximumRetries
Similar to the behavior of Netscape and Internet Explorer.

DO_SetRefreshTimeout
Specifies how long to wait for a meta refresh.

DO_SetRetryWait
Applies to SSL requests. Sets the proxy authorization when accessing SSL pages passed through a proxy server (also known as SSL tunneling).

DO_SetTimeout
Applies to HTTP and SSL requests. Specifies how long to wait for a reply from the server. If a reply is not received within the specified time, the virtual user will fail with a fatal error.

DO_UseEntityList
Applies to HTTP and SSL requests. Decodes non-ASCII character entities.

DO_UseNumericReferenceList
Applies to HTTP and SSL requests. Decodes non-ASCII numeric references.

DO_UsePersistentConnections
Applies to HTTP and SSL requests. Turns the use of persistent connections on or off.

DO_UseProxy
Applies to HTTP and SSL requests. Specifies a proxy server to use during testing.

DO_UseProxyAutomaticConfiguration
Applies to HTTP and SSL requests. Downloads the proxy automatic configuration (PAC) script at the specified URL. The rest of the transaction will use the PAC script to determine which proxy, if any, to connect to hosts.

DO_VerifyDocTitle
Applies to HTTP and SSL requests. Compares the parameters and match type passed in the parameters against the HTML page title specified in the response received from the HTTP request.

DownloadMediaFromASX
Applies to Windows Media Player Streaming Media. Dynamically parses an ASX file from the previous response and initiates and waits for completion of the specified Windows Media resources download.

DownloadMediaRP
Applies to Real Networks Streaming Media. Initiates and waits for completion of the specified multi-media resource download.

DownloadWindowsMedia
Applies to Windows Media Player Streaming Media. Initiates and waits for completion of the specified Windows Media resource download.

EnableStatisticsRP
Applies to Real Networks Streaming Media. Enables capture of media player performance statistics during a load test. Compuware recommends that this function is called in the initial section of a Web script, before the SYNCHRONIZE() call. Although it can be called at any point in the script, this command must appear in the script prior to any DownloadMediaRP call.

Fill_In
Applies to Visual Scripting. Used to represent how the user filled in fields on a form before clicking on a submit button. The values that are passed to Fill_In are expected to be plain text with no encoding other than using + to join multiple selects for LIST_BOX.

Get
Applies to Visual Scripting. Retrieves data from the virtual browser. Whole pages, specific frames, and text strings from within the document can be retrieved.

Navigate_To
Applies to Visual Scripting. Reads a URL typed in the Web browser's address field and constructs a request

to navigate to the URL, or reads another request typed in the browser's address field, finishes the request and navigates to the request. Navigate_To is a direct replacement for DO_Http.

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.

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.

Attach

Applies to Visual Scripting. Changes the current page to the page or frame specified.

This new page becomes the active page that all script commands act on.

Syntax

```
boolean Attach ( 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 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 linkinteger 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);

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 QACenter Performance Edition Installation and Configuration Guide.

Syntax

```
DisableStatisticsRP();
```

Return Value

Parameters

None.

Example

```
DisableStatisticsRP();
```

DO_AddHeader

Applies to HTTP and SSL requests. Indicates headers that are common to every request (DO_Http or DO_Https function calls) in a script.

QALoad takes all of the headers that are in every request in the script and places them at the beginning of the script (between BEGIN_TRANSACTION and the first request) using the DO_AddHeader command. During replay, DO_AddHeader tells QALoad to add the header with a given name and value to all requests in the script.

Note: Cookie and Host headers are not included in the DO_AddHeader function even if they are common to all of the requests.

Syntax

```
DO_AddHeader (const char *name, const char *value);
```

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 Value: TRANSACTION — Clear all temporary variables used in transaction. Value: ALL — Clear everything. Values for Transaction type

Value	Description
ALL_COOKIES	Clear all stored cookies.
BASIC_AUTH_FLAG	Do not send the basic authorization until next challenge.
BAUD_RATE_CALCULATIONS	Clear the accumulated modem emulation data.
CACHE	Clear out any virtual browser cache.
CONNECTION	Reset network connection.
DNS_CACHE	Clear any cached DNS lookups.
INTERNAL_BUFFERS	Clear all internal buffers.
PROXY_AUTH_FLAG	Do not send the basic authorization until next challenge.
REFERER	Clear the referer so it is not sent with the next request.

Values for ALL type

Value	Description
ALL_CGI_PARAMETERS	Clear all Set CGI_PARAMETER parameters.
ALL_HEADERS	Clear all Set HEADER attributes.
ATTACHED_FILES	Clear all files in the binary file list.
BASIC_AUTHORIZATION	Clear the basic authorization username and password.
BLOCK_TRAFFIC_FROM	Clear the block traffic from list.
CERTIFICATE	Clear the client certificate.
CONNECT_REQUEST_FOR_SSL_TUNNELING	Clear the set SSL connect string.
DEFAULT_CONTENT_TYPE	Clear the default content type.
JAVASCRIPT_ENGINE	Clear the JavaScript state.
NTLM_AUTHORIZATION	Clear the NTLM username, password, and domain.
ONLY_ALLOW_TRAFFIC_FROM	Clear the allow traffic from list.
ONLY_USE_SSL_CIPHER	Clear the only use SSL cipher

	string.
PROXY_AUTHORIZATION	Clear the proxy authorization username and password.
PROXY_SETTINGS	Clear all proxy settings.
RECEPTION_BAUD_RATE	Turn off reception baud rate emulation.
SIGNIFICANT_CONTENT_TYPES	Clear the significant content type list.
SPOOFED_IP_ADDRESS	Clear any spoofed IP address.
TRANSMISSION_BAUD_RATE	Turn off transmission baud rate emulation.

Examples

```
DO_Clear ( JAVASCRIPT_ENGINE );
DO_Clear ( CACHE );
DO_Clear ( CONNECTION );
DO_Clear ( ALL_COOKIES );
DO_Clear ( TRANSACTION );
DO_Clear ( ALL );
```

DO_ClearCache

Applies to HTTP and SSL requests. Clears any cached images. Performed automatically by DO_HttpCleanup.

Note: This command is the same as DO_Clear (CACHE).

Syntax

```
DO_ClearCache();
```

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. Also applies to Visual Scripting. Clears memory used by the script.

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

Syntax

```
DO_FreeHttp( );
```

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.

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_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.

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.

Syntax

```
GetFormValueByName( int form_number, const char *field_type, const char *field_name, int
count, char **value );
```

Parameters

Parameter	Description
form_number	Integer specifying which form to search in an HTML document.
field_type	Type of field to search.
field_name	Name of the field to search.
count	If more than one field has the same name, a number specifying each field.
value	Address to a string where the result value will be stored.

Example

```
...
...
char *Field[2];
...
...
for(i=0;i<2;i++)
Field[i]=NULL;
...
...
BEGIN_TRANSACTION();
...
...
DO_GetFormValueByName(FORM (1), "hidden", "hidden", 1, &Field[0]);
DO_GetFormValueByName(FORM (1), "hidden", "hidden1", 1, &Field[1]);
...
...
DO_SetValue("hidden", Field[0]);
DO_SetValue("hidden1", Field[1]);
...
...
BeginCheckpoint(); /* *FORM* */
DO_Http("POST {*action_statement0} HTTP/1.0\r\n"
        "Content-Type: application/x-www-form-urlencoded\r\n"
        "Host: company\r\n"
        "Content-Length: {*content-length}\r\n\r\n"
        "{name }&{hidden}&{hidden1}&{submit}");
...
...
DO_HttpCleanup();
for(i=0; i<2; i++)
{
free(Field[i]);
}
...
...
END_TRANSACTION();
...
...
```

DO_GetHeaderFromReply

Applies to HTTP and SSL requests. Retrieves the value of a header in the reply resulting from a DO_HTTP command.

Syntax

```
int DO_GetHeaderFromReply ( char *header, const char *output_buffer, int nLength )
```

Return Value

1 for success
0 for unsuccessful

Parameters

Parameter	Description
header	A header to look for in the reply.
output_buffer	A string to store the result. Memory should already be allocated for it.
nLength	The length of space available in the output buffer.

Example

```
char OutputBuf[256];
...
...
BEGIN_TRANSACTION();
...
...
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
        "Accept: image/gif, image/x-xbitmap, */*\r\n"
        "Host: company.com\r\n"
        "Cookie: HTMLA=FONTSIZE=LARGE; SITESERVER=ID="
        "4b5b75c9dda4ab9751bce9a95f74ec62\r\n\r\n");
DO_GetHeaderFromReply("Content-Length:", OutputBuf, 255);
...
...
```

DO_GetLastHttpError

Applies to HTTP and SSL requests. Retrieves the integer indicating the error code of the last HTTP request sent with DO_Http.

Errors greater than 399 include the "Page not found" 404 error.

Syntax

```
int DO_GetLastHttpError();
```

Return Value

Returns the error code, or 0 if unsuccessful.

Parameters

None.

Example

```
int error;
char errorString[50];
...
...
BEGIN_TRANSACTION();
...
...
/* Request: 1 */
DO_Http("GET http://company.com/ HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Host: company.com\r\n\r\n");
if ((error = DO_GetLastHttpError()) > 399)
{
sprintf (errorString, "Error in response: %d\n", error);
WWW_FATAL_ERROR ("DO_Http", errorString);
}
...
...
```

DO_GetRedirectedURL

Applies to HTTP requests. Modifies the parameter passed in for use in the next request.

This function is still supported, however, DO_DynamicRedirectHandling is preferred.

Syntax

```
int DO_GetRedirectedURL (char **URL)
```

Return Value

1 for successful
0 for unsuccessful

Parameters

Parameter	Description
URL	An address to a string.

Example

```
DO_Http(http_statement);

/* RedirectedURL[0]="http://company/cgi-bin/pm3D.htm"*/
DO_GetRedirectedURL(&RedirectURL[0]);

/* Request: 10 * From: QALoad WWW Capture Examples */
DO_SetValue("redirect_statement0", RedirectURL[0]);
DO_Http("GET {*redirect_statement0} HTTP/1.0\r\n"
        "Accept: */*\r\n"
        "Referer: http://company/index.htm\r\n"
        "Accept-Language: en-us\r\n"
        "Accept-Encoding: gzip, deflate\r\n"
        "Host: company\r\n\r\n");
```

DO_GetReplyBuffer

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

Syntax

```
Const char * DO_GetReplyBuffer()
```

Return Value

The last HTTP reply or NULL if unsuccessful.

Parameters

None.

Example

```
const char * data;
...
...
BEGIN_TRANSACTION();
...
...

/* Request: 1 */

DO_Http("GET http://company.com/ HTTP/1.0\r\n\r\n" );
data = strstr(DO_GetReplyBuffer(), "data_key" );
if ( data == NULL )
{
WWW_FATAL_ERROR ("DO_Http", "Data_key was missing in reply" );
}
...
...
```

DO_GetUniqueString

Applies to HTTP and SSL requests. Used to parse the most recent HTTP server reply to get the contents of a string that occurs between the left and right input strings.

Syntax

```
char *DO_GetUniqueString( const char *left, const char *right );
```

Return Value

The string (null-terminated) of characters between the left and right search strings provided as input. NULL If either the left or right search strings are not found.

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_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

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>version</td><td>WWWHTTPVersionEnum
The HTTP version. If specified as Auto, the version used for each request is determined from the request. Valid values are:
<table><tr><th>Value</th><th>Description</th></tr><tr><td>"1.0"</td><td>HTTP version 1.0</td></tr><tr><td>"1.1"</td><td>HTTP version 1.1</td></tr><tr><td>"AUTO"</td><td>HTTP version is set in DO_Http and DO_Https</td></tr></table></td></tr>
</table>

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 NTLM authentication.

DO_NTLMAuthorization is related to the NTLM option on the QALoad Script Development Workbench Record Options wizard. When you select that option and enter user ID and password information, DO_NTLMAuthorization(string, string) is written to your script. QALoad attempts to use the user ID and password you entered to access the site. If the information is not accepted, QALoad reports the error and aborts.

At test time, when QALoad encounters a NTLM controlled site, it uses the NTLM user ID and password that are provided to access that site.

Note: NTLM user names and passwords can be variablized by machine, but not by user.

Syntax

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

Return Value

Parameters

Parameter	Description
name	A valid user ID for the NTLM-enabled site.
password	A valid password corresponding to the user ID.

Examples

Example 1:

```
...
...
BEGIN_TRANSACTION();
...
...
DO_NTLMAuthorization("user-id", "~encr~2038520348AKJAS");
...
...
END_TRANSACTION();
...
...
```

Note: String must be enclosed in quotation marks (""), unless NULL is used.

Example 2:

When the user ID, password, and domain are provided:

```
DO_NTLMAuthorization("domain\\user_id","~encr~506C205A545D");
```

Example 3:

When the domain is not provided:

```
DO_NTLMAuthorization("user_id","~encr~506C205A545D");
```

Example 4:

When NULL is used and access is provided:

```
DO_NTLMAuthorization(NULL, NULL);
```

Note: NULL is not enclosed in quotes.

DO_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 WWW Advanced 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_SetCheckpointName

Sets the name of the next automatic checkpoint for the next DO_Http or DO_Https statement in the script.

Syntax

```
void DO_SetCheckpointName (const char *szCheckpointName);
```

Return Value

Parameters

Parameter	Description
szCheckpointName	The name for the next automatic checkpoint.

Example

```
DO_SetCheckpointName("Login to Website");
DO_Https("POST https://dbhost.company.com/login.asp HTTP/1.1\r\n"
    "\r\n"
    "{domain}&{username}&{password}");
```

DO_SetCookie

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

The path of the cookie is "/". The domain of the cookie is the same as the next DO_Http or DO_Https request. If you wish to set a particular domain or path, use DO_SetCookieEx.

Once a cookie is set, it remains for the rest of the transaction. To remove the cookie, use DO_SetCookieEx with the name of the cookie to remove and an expiration value of -1.

DO_SetCookie requires DO_DynamicCookieHandling to be set to TRUE.

Syntax

```
BOOL DO_SetCookie ( const char * szName, const char * szValue );
```

Return Value

TRUE for successful
FALSE for unsuccessful

Parameters

Parameter	Description
szName	Name of the cookie to set.
szValue	Value of the cookie to set.

Example

```
...
...
BEGIN_TRANSACTION();
...
...
DO_SetCookie ( "cookie1", "desired value" );
/* Request: 1 */
```

```
/*
* This request will have "cookie1" sent with this request
*/
DO_Http ( "GET http://company.com/ HTTP/1.0\r\n\r\n" );
...
...
```

DO_SetCookieEx

Applies to HTTP and SSL requests. DO_SetCookie adds a cookie to the current transaction.

Once a cookie is set, it remains for the rest of the transaction. To remove the cookie, use DO_SetCookieEx with the name of the cookie to remove and a max age of -1.

DO_SetCookie requires DO_DynamicCookieHandling to be set to TRUE.

Syntax

```
BOOL DO_SetCookieEx ( const char * szName, const char * szValue,
                      const char * szDomain, const char * szPath,
                      int nMaxAge, BOOL bSecure );
```

Return Value

TRUE for successful
FALSE for unsuccessful

Parameters

Parameter	Description
szName	Name of the cookie to set.
szValue	Value of the cookie to set.
szDomain	Domain of the cookie. The domain of the cookie controls what hosts the cookie is sent to.
szPath	The path of the cookie. The path of the cookie controls when a cookie is sent to a host based on the path of the URL.
nMaxAge	Time to live of the cookie. Use a value of 0 for a session cookie and -1 for an expired cookie.
bSecure	Boolean flag (TRUE or FALSE). If the value is TRUE, then the cookie only is sent with SSL request. If the value is FALSE, then the cookie is sent with HTTP and SSL requests.

Example

```
...
...
BEGIN_TRANSACTION();
...
...
DO_SetCookieEx ( "cookie1", "desired value", ".company.com", "/", 1000, FALSE );

/* Request: 1 */
/*
* This request will have "cookie1" sent with this request
*/
```

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

DO_SetJavascriptCleanupThreshold

Applies to HTTP and SSL requests. Periodically QALoad destroys its internal JavaScript model and recreates it.

DO_SetJavascriptCleanupThreshold sets a count of the number of times JavaScript parsing is done before destroying and recreating the model. By default, the count is 300.

Cleaning up JavaScript takes CPU time, and the Javascript model takes up more memory the longer the same model is used. To reduce CPU usage, set the count higher. To reduce the memory footprint, set the count lower.

Syntax

```
DO_SetJavascriptCleanupThreshold(int nThreshold)
```

Return Value

Parameters

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

Example

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

DO_SetMaxBrowserThreads

Applies to HTTP and SSL requests. Specifies the number of concurrent connections to make for playback.

This command relates to the Max Concurrent Connections option on the WWW Advanced options dialog box. The value you enter in that field is inserted in the script.

Syntax

```
DO_SetMaxBrowserThreads(int count);
```

Return Value

Parameters

Parameter	Description
count	The number of connections to make. *QA*Load accepts 1-8. The default is 2.

Example

```
BEGIN_TRANSACTION();
DO_SetMaxBrowserThreads(2);
```

DO_SetMaximumRetries

Applies to HTTP and SSL requests. Sets the maximum number of times a virtual user should attempt to retrieve a graphic or page that failed.

Similar to the behavior of Netscape and Internet Explorer.

Syntax

```
DO_SetMaximumRetries(int nValue)
```

Return Value

Parameters

Parameter	Description
nValue	The default is 4.

Example

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

DO_SetRefreshTimeout

Specifies how long to wait for a meta refresh or an HTTP refresh header.

The HTML meta tag can set a number of seconds before a refresh. When that number of seconds has expired, then the browser loads the URL specified in the meta refresh.

QALoad's WWW replay only refreshes the page if the number of seconds specified in the refresh is less than or equal to the timeout value set by DO_SetRefreshTimeout. If the refresh is set too large, then QALoad's WWW replay can get stuck in an infinite loop.

Syntax

```
int DO_SetRefreshTimeout(int nTimeout);
```

Parameters

Parameter	Description
nTimeout	How many seconds to wait for a refresh, the default is 0.

DO_SetRetryWait

Applies to HTTP and SSL requests. Sets the delay between retries in seconds.

Syntax

```
DO_SetRetryWait(int nValue)
```

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 QALoad 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 ) ;
```

DownloadMediaFromASX

Applies to Windows Media Player streaming media.

Dynamically parses an ASX file from the previous response and initiates and waits for completion of the specified Windows Media resources download.

DownloadMediaFromASX is a deprecated function. Use a combination of the Click_On function with the PlayMedia function instead.

Note: For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad" chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

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

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 *QA*Center Performance Edition 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);
DownloadMediaRP("http://host2:8080/realmp3.mp3", 10);
```

DownloadMediaWMP

Applies to Windows Media Player streaming media. Initiates and waits for completion of the specified Windows Media resource download.

DownloadMediaWMP is a deprecated function. Use a combination of the Click_On function with the PlayMedia function instead.

Note: For streaming media playback, QALoad requires specific media player versions. For a list of supported versions, refer to "System Requirements" in the "Installing QALoad " chapter of the QACenter Performance Edition Installation and Configuration Guide.

Syntax

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

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 *QA*Center Performance Edition Installation and Configuration Guide.

Syntax

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

Return Value

Parameters

Parameter	Description
`flags`	Determines which statistics to show. The flag values in the following table can be combined using a logical OR. Flag values include: QAL_WWW_RN_STAT_ALL_LEVELS: All statistic levels QAL_WWW_RN_STAT_PLAYER: Media Player level statistics QAL_WWW_RN_STAT_SOURCE: Source level statistics QAL_WWW_RN_STAT_STREAM: <not implemented> QAL_WWW_RN_STAT_ALL: Enable all levels, all counters QAL_WWW_RN_STAT_PLAYER_ALL: All Media Player level statistics QAL_WWW_RN_STAT_SOURCE_ALL: All source level statistics QAL_WWW_RN_STAT_STREAM_ALL: All stream level statistics QAL_WWW_RN_STAT_ALL_COUNTERS: All counters QAL_WWW_RN_STAT_NORMAL_PKTS: Packets not lost, late, etc. QAL_WWW_RN_STAT_RECOVERED_PKTS: Packets recovered QAL_WWW_RN_STAT_RECEIVED_PKTS: Packets received QAL_WWW_RN_STAT_LOST_PKTS: Packets currently lost QAL_WWW_RN_STAT_LATE_PKTS: Late packets QAL_WWW_RN_STAT_CLIP_BANDWIDTH: Bandwidth at which the clip was encoded QAL_WWW_RN_STAT_AVE_BANDWIDTH: Average bandwidth so far QAL_WWW_RN_STAT_CUR_BANDWIDTH: Current bandwidth
`interval`	`Report every nth stat received.`
`traceOutput`	`TRUE means send enabled stats to QALoad Player window (if QALoad Player window output is enabled).`

Example

```
// Records, current bandwidth, average bandwidth, and the clip
// bandwidth at the Player (media player) level as often as
// the statistics are updated.

EnableStatisticsRP( QAL_WWW_RN_STAT_PLAYER
                    QAL_WWW_RN_STAT_AVE_BANDWIDTH
                    QAL_WWW_RN_STAT_CLIP_BANDWIDTH
                    QAL_WWW_RN_STAT_CUR_BANDWIDTH,
                    0, TRUE );
```

Fill_In

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

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);

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 QACenter Performance Edition 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 );

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 );
```

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 QACenter Performance Edition 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
```

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");
```

Error Codes

Citrix

Citrix Playback Error Codes

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

Error code	Description
CTX_ERROR_00001	CoInitialize failed
CTX_ERROR_00002	Unable to attach tls to window
CTX_ERROR_00003	Unable to clear properties
CTX_ERROR_00004	Unable to set scaling mode
CTX_ERROR_00005	Unable to set initial program
CTX_ERROR_00006	Unable to set working directory
CTX_ERROR_00007	Unable to set username
CTX_ERROR_00008	Username is NULL
CTX_ERROR_00009	Unable to set password
CTX_ERROR_00010	Password is NULL
CTX_ERROR_00011	Unable to set domain
CTX_ERROR_00012	Invalid argument to CitrixConnect
CTX_ERROR_00013	Could not load ICA File: <Loading ICA File>
CTX_ERROR_00014	Could not find ICA File: <Loading ICA File>
CTX_ERROR_00015	Connect timeout expired
CTX_ERROR_00016	Wait for connect abandoned
CTX_ERROR_00017	Disconnect timeout expired
CTX_ERROR_00018	Wait for disconnect abandoned
CTX_ERROR_00022	Invalid keyboard entry

CTX_ERROR_00023	Unable to move mouse
CTX_ERROR_00024	Unable to click mouse
CTX_ERROR_00025	Unable to send ping
CTX_ERROR_00026	Ping timeout reached
CTX_ERROR_00027	Wait for ping abandoned
CTX_ERROR_00028	Window <Window Name> does not exist
CTX_ERROR_00029	Unable to put address
CTX_ERROR_00030	Unable to unmarshal <session>
CTX_ERROR_00031	Sending ascii key <key> failed
CTX_ERROR_00032	Sending keydown <key> failed
CTX_ERROR_00033	Unable to bring window to top
CTX_ERROR_00034	Unexpected event: expected create <Window>
CTX_ERROR_00035	Unexpected event: expected destroy <Window>
CTX_ERROR_00036	Unexpected event: expected <action> <window> action
CTX_ABORT_00037	The Citrix client has unexpectedly terminated <WaitForWindowCreate>. Script Aborting
CTX_ERROR_00038	Unable to marshal session pointer
CTX_ERROR_00039	Unable to marshal keyboard pointer
CTX_ERROR_00040	Unable to marshal mouse pointer
CTX_ERROR_00041	Unable to register for session events
CTX_ERROR_00042	Unable to register for keyboard events
CTX_ERROR_00043	Unable to register for mouse events
CTX_ERROR_00044	Unable to set connect event
CTX_ERROR_00045	Unable to set ping event
CTX_ERROR_00046	Unable to connect
CTX_ERROR_00047	Unable to disconnect
CTX_ERROR_00048	Unable to logon

CTX_ERROR_00049	Unable to logoff
CTX_ERROR_00050	Unable to use ICA file
CTX_WARNING_00051	Disconnect timeout expired
CTX_WARNING_00052	Disconnect timeout abandoned
CTX_WARNING_00053	Unable to logoff
CTX_WARNING_00054	Unable to disconnect
CTX_ERROR_00060	Unable to execute statement
CTX_ERROR_00061	Unable to unmarshal <keyboard>
CTX_ERROR_00062	Unable to unmarshal <mouse>

CTX_ERROR_00001

Unable to initialize the COM library.

Description:

The COM library is used to communicate with the Citrix client during replay.

Script Commands:

CitrixInit

Causes:

- The Citrix client or QALoad were not successfully installed.
- The replay machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

Verify the installations of Citrix client and QALoad were successful and re-install if not.

External Sources:

None

CTX_ERROR_00002

The QALoad Citrix replay code is unable to initialize correctly.

Description:

The QALoad Citrix replay code was unable to initialize the Citrix client code for use with the replay script.

Script Commands:

CitrixInit

Causes:

- QALoad was not successfully installed.
- The client machine is low on resources.

Actions:

- Verify the QALoad installation was successful and re-install if not.
- Verify the client machine has sufficient resources.

External Sources:

None

CTX_ERROR_00003

Unable to clear the properties in Citrix API COM object.

Description:

The Citrix client was unable to initialize the COM properties. This may indicate a problem with the Citrix environment.

Script Commands:

CitrixInit

Causes:

- The Citrix client or QALoad were not successfully installed.
- The replay machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

- Verify the installations of Citrix client and QALoad were successful and re-install if not.
- Verify the client machine has sufficient resources.
- Verify the integrity of the installation of the OS on the replay machine.

External Sources:

None

CTX_ERROR_00004

Unable to clear the properties in Citrix API COM object.

Description:

The Citrix client was unable to set the scaling mode for the client. This may indicate a problem with the Citrix environment.

Script Commands:

CitrixInit

Causes:

- The Citrix client or QALoad were not successfully installed.
- The replay machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

- Verify the installations of Citrix client and QALoad were successful and re-install if not.
- Verify the client machine has sufficient resources.
- Verify the integrity of the installation of the OS on the replay machine.

External Sources:

None

CTX_ERROR_00005

Unable to set the initial program for the Citrix client session.

Description:

Setting the initial program in the Citrix client returned an error.

Script Commands:

CtxConnect

Causes:

The initial program does not exist or is not in the specified directory.

Actions:

Ensure the initial program exists, is in the correct directory, and is being called correctly.

External Sources:

None

CTX_ERROR_00006

Unable to set the initial directory for the Citrix client session.

Description:

Setting the start-up directory for the Citrix session returned an error.

Script Commands:

CtxConnect

Causes:

The specified directory does not exist.

Actions:

Ensure the specified directory exists.

External Sources:

None

CTX_ERROR_00007

Unable to set the username.

Description:

Setting the username for the Citrix session returned an error.

Script Commands:

CtxSetLoginInfo

CtxDomainLoginInfo

CtxConnect

Causes:

The username is not a valid username for the Citrix session

Actions:

Ensure the username specified is valid for the Citrix session.

External Sources:

None

CTX_ERROR_00008

There is no username specified for the Citrix session.

Description:

The Citrix session returned an error because no username was specified for the session.

Script Commands:

CtxSetLoginInfo

CtxDomainLoginInfo

Causes:

No username was specified for the Citrix client connection.

Actions:

Ensure a valid username is specified for the Citrix session.

External Sources:

None

CTX_ERROR_00009

Unable to set the password.

Description:

Setting the password for the Citrix session returned an error.

Script Commands:

CtxSetLoginInfo

CtxDomainLoginInfo

CtxConnect

Causes:

The password is not valid for the username specified in the Citrix session

Actions:

Ensure both the username and password specified are valid for the Citrix session.

External Sources:

None

CTX_ERROR_000010

There is no password specified for the Citrix session.

Description:

The Citrix session returned an error because no password was specified for the username in the session.

Script Commands:

CtxSetLoginInfo

CtxDomainLoginInfo

Causes:

No password was specified for the username in the Citrix client connection.

Actions:

Ensure a valid username and password are specified for the Citrix session.

External Sources:

None

CTX_ERROR_00011

The domain specified for the Citrix session is invalid.

Description:

The Citrix session returned an error because the domain specified was invalid for the Citrix session.

Script Commands:

CtxSetLoginInfo

CtxDomainLoginInfo

CtxConnect

Causes:

The domain specified is not valid for the Citrix session.

Actions:

Ensure the domain specified in the connection is valid for the Citrix session.

External Sources:

None

CTX_ERROR_00012

An argument supplied to the Citrix connect call is invalid.

Description:

The Citrix connect call failed because an argument specified as a parameter to the call is invalid.

Script Commands:

CtxConnect

Causes:

An argument specified to the Citrix connect call

Actions:

Ensure that the parameters passed to the CtxConnect call are correct.

External Sources:

None

CTX_ERROR_00013

The specified Citrix ICA file cannot be loaded.

Description:

The Citrix client returned an error when it tried to load the Citrix ICA specified.

Script Commands:

CtxConnect

Causes:

The Citrix ICA file specified is not a valid ICA file.

Actions:

Ensure that the specified Citrix ICA file is valid by connecting to a Citrix session using that file in Citrix Neighborhood.

External Sources:

None

CTX_ERROR_00014

The specified Citrix ICA file cannot be found.

Description:

The ICA file specified could not be found by the Citrix client session.

Script Commands:

CtxConnect

Causes:

The file path name of the ICA file specified points to an non-existent file.

Actions:

Identify the correct CIA file and path and ensure that it is specified correctly in the Citrix options and/or in the script.

External Sources:

None

CTX_ERROR_00015

The Citrix client cannot connect to the server in the time specified in the Citrix options.

Description:

The connection was not made in the time interval specified in the script and/or the Citrix options dialog.

Script Commands:

CtxConnect

Causes:

- The Citrix server is not online and receiving connections.
- The timeout specified in the script or in the Citrix options dialog is too low.

Actions:

- Ensure that the Citrix server is available for connection.
- Change the Connection Timeout value to a higher value (in seconds).
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00016

The attempt to connect to the Citrix was abandoned.

Description:

The connection was not made in the time interval before the Citrix client abandoned the attempt.

Script Commands:

CtxConnect

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow connections.

Actions:

- Ensure that the Citrix server is available for connection.
- Change the Connection Timeout value to a higher value (in seconds).
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00017

The Citrix client cannot disconnect to the server in the time specified in the Citrix options.

Description:

The connection was not closed in the time interval specified in the script and/or the Citrix options dialog.

Script Commands:

CitrixUninit

CtxDisconnect

Causes:

- The Citrix server went offline due to a server error and could not drop the connection.
- The timeout specified in the script or in the Citrix options dialog is too low.

Actions:

- Ensure that the Citrix server is functional.
- Ensure that the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00018

The call to the Citrix API disconnect event was abandoned.

Description:

The disconnection response was not received before the Citrix client abandoned the attempt.

Script Commands:

CitrixUninit

CtxDisconnect

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00022

The keyboard input is not correct.

Description:

The keyboard input is not correct for the keyboard locale on the replay machine.

Script Commands:

CtxType

CtxTypeVK

Causes:

The Citrix server could not process the keyboard entry made for the keyboard locale for the Citrix server.

Actions:

- Ensure the Citrix server is using the same keyboard locale as the Citrix client.
- Ensure the keyboard locale of the server has not changed since the script was recorded.

External Sources:

None

CTX_ERROR_00023

The call to the Citrix API to move the mouse failed.

Description:

The call to the Citrix API returned a failure and could not perform the mouse move action.

Script Commands:

CtxMouseMove

Causes:

- The API call is to an invalid or non-existent window.
- The move coordinates are not valid for the Citrix desktop.

Actions:

Add a call to the script to see that the window exists. Refer to Handling Dynamic Windows.

External Sources:

None

CTX_ERROR_00024

The call to the Citrix API to perform a mouse click action failed.

Description:

The call to the Citrix API returned a failure and could not perform the mouse click action.

Script Commands:

CtxClick

CtxDoubleClick

Causes:

- The API call is to an invalid or non-existent window.
- The mouse is not over the window specified in the API call.

Actions:

- Add a call to the script to see that the window exists. Refer to Handling Dynamic Windows.
- Add a mouse move action prior to the mouse click call that places the coordinates in the correct location for the mouse click action.

External Sources:

None

CTX_ERROR_00025

The call to the Citrix API to perform a ping action failed.

Description:

The call to the Citrix API returned a failure and could not perform the ping action.

Script Commands:

CtxPing

Cause:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00026

The call to the Citrix API to perform a ping action timed out.

Description:

The Citrix API call to perform the ping action did not receive a response from the server within the timeout specified in the script and/or the Citrix options.

Script Commands:

CtxPing

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.

- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00027

The call to the Citrix API to perform a ping action was abandoned by the Citrix client.

Description:

The Citrix API call to perform the ping action did not receive a response from the server.

Script Commands:

CtxPing

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00028

The window specified in the API call does not exist.

Description:

The window specified as a parameter to the API call has not been created or has been destroyed and the API call cannot perform the action.

Script Commands:

CtxTypeVK CtxType

CtxType CtxMouseDown

CtxKeyDown CtxMouseUp

CtxKeyUp CtxClick

CtxTypeVK CtxDoubleClick

Causes:

- The API call is to an invalid or non-existent window.
- The window has not been created yet or has been destroyed.

Actions:

Add a call to the script to see that the window exists. Refer to Handling Dynamic Windows.

External Sources:

None

CTX_ERROR_00029

The Citrix API is not able to process the host address.

Description:

The Citrix API call returned an error when processing the host address.

Script Commands:

CtxConnect

Causes:

The host address specified in the API call is not a valid Citrix server.

Actions:

Ensure that the host address specified in the CtxConnect call is valid.

External Sources:

None

CTX_ERROR_00030

The Citrix COM object is unable to free the marshaled data for the session.

Description:

The Citrix client was unable to free the COM session data. This is usually due to a problem with the Citrix COM client.

Script Commands:

CtxConnect

Causes:

- The client machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

- Verify that the client machine has sufficient resources.
- Verify the integrity of the client machine installations.

External Sources:

None

CTX_ERROR_00031

The Citrix client cannot send the key press to the specified window.

Description:

The Citrix client returned an error when it tried to send the key to the window in the KeyPress API call.

Script Commands:

CtxType

Causes:

- The key press event does not have a window to process the event.
- The specified key is invalid.

Actions:

Ensure that the correct key code is used for the keyboard settings used by the client.

External Sources:

None

CTX_ERROR_00032

The Citrix client cannot send the key down to the specified window.

Description:

The Citrix client returned an error when it tried to send the key down to the window in the KeyDown API call.

Script Commands:

CtxKeyDown

Causes:

- The key down event does not have a window to process the event.
- The specified key is invalid.

Actions:

Ensure that the correct key code is used for the keyboard settings used by the client.

External Sources:

None

CTX_ERROR_00033

The Citrix client cannot bring the window to the foreground.

Description:

The Citrix client returned an error from the call to bring the specified window to the foreground.

Script Commands:

CtxTypeVK	CtxType
CtxType	CtxMouseDown
CtxKeyDown	CtxMouseUp
CtxKeyUp	CtxClick
CtxTypeVK	CtxDoubleClick

Causes:

- The window specified does not exist.
- The window could not be brought to the foreground because a modal window is wafting for an event.

Actions:

Ensure that there isn't a window wafting for an event at this point in the script. If so, insert script commands to process that window prior to this call.

External Sources:

None

CTX_ERROR_00034

The Citrix client timed out waiting for a CreateWindow event.

Description:

The Citrix client did not receive a CreateWindow event for the specified window within the time allocation value specified in the script and/or the Citrix options.

Script Commands:

Causes:

- The specified window is created intermittently during the Citrix session.
- Another window did not get an event processed and as a result, this window create event did not occur.

Actions:

Validate the script to see that the window is created consistently at this point in the script. If not, add script commands to conditionally check for the window creation event. Refer to Handling Dynamic Windows.

External Sources:

None

CTX_ERROR_00035

The Citrix client timed out waiting for a DestroyWindow event.

Description:

The Citrix client did not receive a DestroyWindow event for the specified window within the time allocation value specified in the script and/or the Citrix options.

Script Commands:

CtxWaitForWindowDestroy

Causes:

- The specified window is created intermittently during the Citrix session.
- Another window did not get an event processed and as a result, this window destroy event did not occur.

Actions:

Validate the script to see that the window is created consistently at this point in the script. If not, add script commands to conditionally check for the window creation event. (See the Advanced Scripting Guide)

External Sources:

None

CTX_ERROR_00036

The Citrix client timed out waiting for the specified event.

Description:

The Citrix client was not notified of this event for the specified window within the time allocation value specified in the script and/or the Citrix options.

Script Commands:

CtxWaitForWindowCreate
CtxWaitForWindowMove
CtxWaitForWindowDestroy
CtxWaitForWindowResize
CtxWaitForWindowEventExists
CtxWaitForWindowStyleChange
CtxWaitForCaptionChange
CtxWaitForScreenUpdate
CtxWaitForWindowMinimize

Causes:

- The specified window is created intermittently during the Citrix session.
- Another window did not get an event processed and as a result this window destroy event did not occur.

Actions:

Validate the script to see that the window is created consistently at this point in the script and that the session state is consistent with the event actions. If not, add script commands to ensure the session state is consistent with the expected event. Refer to Handling Dynamic Windows.

External Sources:

None

CTX_ERROR_00037

The Citrix client terminated unexpectedly.

Description:

The Citrix client has unexpectedly terminated for an unknown reason and the script cannot process commands.

Script Commands:

CtxWaitForWindowCreate
CtxWaitForWindowMove
CtxWaitForWindowDestroy
CtxWaitForWindowResize
CtxWaitForWindowEventExists
CtxWaitForWindowStyleChange

CtxWaitForCaptionChange CtxWaitForScreenUpdate

CtxWaitForWindowMinimize

Causes:

- The replay machine is low on resources.
- The network could not process the traffic from the client to the server.
- The replay machine is in an unknown state.

Actions:

- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00038

The Citrix client cannot marshal the session resources.

Description:

The Citrix COM client could not allocate the marshal resources to process session objects and events.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00039

The Citrix client cannot marshal the keyboard resources.

Description:

The Citrix COM client could not allocate the marshal resources to process keyboard objects and events.

Script Commands:

CtxWaitForWindowCreate CtxWaitForWindowMove

CtxWaitForWindowDestroy CtxWaitForWindowResize

CtxWaitForWindowEventExists CtxWaitForWindowStyleChange

CtxWaitForCaptionChange CtxWaitForScreenUpdate

CtxWaitForWindowMinimize

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00040

The Citrix client cannot marshal the mouse resources.

Description:

The Citrix COM client could not allocate the marshal resources to process mouse objects and events.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00041

The player cannot register with the Citrix client for session event messages.

Description:

The call to the Citrix client to register for session events failed.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.

- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00042

The player cannot register with the Citrix client for keyboard event messages.

Description:

The call to the Citrix client to register for keyboard events failed.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00043

The player cannot register with the Citrix client for mouse event messages.

Description:

The call to the Citrix client to register for mouse events failed.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.

- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00044

The player cannot set the connect event with the Citrix client.

Description:

The call to the Citrix client to register for the connection event failed.

Script Commands:

CitrixInit

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. (Reboot if necessary)

External Sources:

None

CTX_ERROR_00045

The player cannot set the ping event with the Citrix client.

Description:

The call to the Citrix client to register for the ping event failed.

Script Commands:

CtxPing

Causes:

- QALoad or the Citrix client are not installed properly.
- The replay machine is not in a consistent state.

Actions:

- Ensure the integrity of the QALoad and Citrix client installations.
- Ensure the replay machine has sufficient resources to replay the number of virtual users.
- Ensure the replay machine is in a functional state. Reboot if necessary.

External Sources:

None

CTX_ERROR_00046

The Citrix client cannot connect to the Citrix server.

Description:

The call to initiate a connection with the Citrix server failed.

Script Commands:

CtxConnect

Causes:

- The host address specified in the API call is not a valid Citrix server.
- The Citrix server is not available.

Actions:

Ensure that the parameters passed to the CtxConnect call are correct.

External Sources:

None

CTX_ERROR_00047

The Citrix client cannot disconnect from the Citrix server.

Description:

The call to disconnect from the Citrix server failed.

Script Commands:

CtxDisconnect

Causes:

The Citrix server is not processing requests from the Citrix client.

Actions:

Ensure that the Citrix server is up and able to process requests.

External Sources:

None

CTX_ERROR_00048

The Citrix client cannot log on to the Citrix server.

Description:

The Citrix client failed to log on to the Citrix server.

Script Commands:

CtxConnect

Causes:

The username or password set for the session are not valid for the Citrix server.

Actions:

- Ensure the username and password are correct for the Citrix server.
- Ensure the Citrix server can accept new sessions.

External Sources:

None

CTX_ERROR_00049

The Citrix client cannot log off the Citrix server.

Description:

The Citrix client failed to log off the Citrix server.

Script Commands:

CtxConnect

Causes:

The Citrix server is not processing requests from the Citrix client.

Actions:

Ensure that the Citrix server is up and able to process requests.

External Sources:

None

CTX_ERROR_00050

The ICA file specified in the script is not valid.

Description:

The Citrix client could use the specified ICA to connect and log on to the Citrix server.

Script Commands:

CtxConnect

Causes:

The wrong ICA file was specified, or the specified file is not a valid ICA file.

Actions:

- Ensure that the ICA file is correct in the script.
- Verify the ICA file is correct by connecting to the Citrix server using that file in Citrix Neighborhood.

External Sources:

None

CTX_WARNING_00051

The Citrix client cannot disconnect to the server in the time specified in the Citrix options.

Description:

The connection was not closed in the time interval specified in the script and/or the Citrix options dialog.

Script Commands:

CtxDisconnect

Causes:

- The Citrix server went offline due to a server error and could not drop the connection.
- The timeout specified in the script or in the Citrix options dialog is too low.

Actions:

- Ensure that the Citrix server is functional.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_WARNING_00052

The call to the Citrix API disconnect event was abandoned.

Description:

The disconnection response was not received before the Citrix client abandoned the attempt.

Script Commands:

CtxDisconnect

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_WARNING_00053

The Citrix client cannot log off the Citrix server.

Description:

The Citrix client failed to log off the Citrix server.

Script Commands:

CtxDisconnect

Causes:

The Citrix server is not processing requests from the Citrix client.

Actions:

Ensure that the Citrix server is up and able to process requests.

External Sources:

None

CTX_WARNING_00054

The call to the Citrix API disconnect event was abandoned.

Description:

The disconnection response was not received before the Citrix client abandoned the attempt.

Script Commands:

CtxDisconnect

Causes:

- The Citrix server is not online and receiving connections.
- The network is too slow to allow communication with the client.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_Error_00060

The Citrix client has already disconnected, so the statement could not be executed.

Description:

The Citrix client has already disconnected from the server, so the action cannot be performed.

Script Commands:

CtxWaitForWindowCreate
CtxWaitForWindowDestroy
CtxWaitForWindowActivate
CtxWaitForCaptionChange
CtxWaitForWindowMinimize
CtxWaitForWindowMove
CtxWaitForWindowResize
CtxWaitForWindowStyleChange
CtxWaitForScreenUpdate
CtxKeyDown
CtxKeyUp
CtxTypeVK
CtxType
CtxMouseDown
CtxMouseUp
CtxClick
CtxDoubleClick

Causes:

- The Citrix server is not online and receiving connections.

- The network is too slow to allow communication with the client.
- The script may have disconnected as a result of a change in behavior from the recorded session.

Actions:

- Ensure that the Citrix server is available.
- Ensure the network is not overloaded with other traffic.

External Sources:

None

CTX_ERROR_00061

The Citrix COM object is unable to free the marshaled data for the keyboard.

Description:

The Citrix client was unable to free the COM keyboard data. This is usually due to a problem with the Citrix COM client.

Script Commands:

CtxConnect

Causes:

- The client machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

- Verify the client machine has sufficient resources.
- Verify the integrity of the client machine installations.

External Sources:

None

CTX_ERROR_00062

The Citrix COM object is unable to free the marshaled data for the mouse.

Description:

The Citrix client is unable to free the COM mouse data. This is usually due to a problem with the Citrix COM client.

Script Commands:

CtxConnect

Causes:

- The client machine is low on resources.
- Windows installation on the replay machine is incorrect.

Actions:

- Verify the client machine has sufficient resources.
- Verify the integrity of the client machine installations.

External Sources:

None

Oracle Forms Server

Oracle Forms Server Playback Error Codes

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

Note: Most of the errors listed below are client request errors related to JVM memory issues. When the error is due to a server problem, the error message indicates a connection issue or a bad response from the server. All these errors cause playback to fail. When the error is client-related, you can work around the JVM memory issue by tweaking the Player machine's Threads Per Player value in QALoad Conductor. When the error is server-related, the server is unable to handle the load. The server typically throws out connection requests, does not respond to requests, or terminates connections during playback.

Error code	Description
OFS_ERROR_00001	Failed to allocate buffer for installation path
OFS_ERROR_00002	Failed to load loadplayerJava.dll
OFS_ERROR_00003	Exception in function <function >: <exception>
OFS_ERROR_00004	Failed to send a heartbeat message
OFS_ERROR_00101	Unable to connect to server
OFS_ERROR_00102	Unable to communicate with server
OFS_ERROR_00103	Unable to disconnect
OFS_ERROR_00104	OracleAppsLogin: Error: icx_ticket not found in OracleAppsLogin, please check URL, userID and password
OFS_ERROR_00105	Connection terminated by server
OFS_ERROR_00106	Server did not return the encryption keys
OFS_ERROR_00107	Server reply data is invalid
OFS_ERROR_00108	Failed to get the content length of the POST request
OFS_ERROR_00109	Failed to get the reply content
OFS_ERROR_00110	Invalid URL
OFS_ERROR_00111	Failed to store data from the server reply
OFS_ERROR_00112	The server sent an error message

OFS_ERROR_00113	Failed while reading the server reply
OFS_ERROR_00114	SSL Handshake failed
OFS_ERROR_00115	Failed to close SSL socket
OFS_ERROR_00116	Failed while processing server detail message. Unknown control handle
OFS_ERROR_00117	ICX Ticket not found in OracleAppsLogin
OFS_ERROR_00118	Failed to load loadplayerJava library at startup
OFS_ERROR_00119	Failed to create the replay log file
OFS_ERROR_00120	Unable to write to replay capture file
OFS_ERROR_00121	JVM memory issues
OFS_ERROR_00122	Invalid argument: {}
OFS_ERROR_00123	Internal error

OFS_ERROR_00001

Failed to allocate buffer for installation path.

Description:

The QALoad Player ran out of memory during test initialization.

Script Commands:

N/A

Causes:

The replay machine is low on resources (memory).

Actions:

Verify the client machine has sufficient resources.

External Sources:

None

OFS_ERROR_00002

Failed to load loadplayerJava.dll

Description:

OFS replay was unable to load QALoad dll loadplayerJava.

Script Commands:

ofsSetRunOptions

Causes:

The dll is missing or corrupt.

Actions:

Verify the QALoad installation was successful and re-install if not.

External Sources:

None

OFS_ERROR_00003

Exception in function {}: {}

Description:

An unhandled exception was encountered in the specified function.

Script Commands:

All

Causes:

An exception was not caught by the OFS middleware.

Actions:

See addition information provided with this message.

External Sources:

None

OFS_ERROR_00004

Can not find method {}

Description:

OFS replay is unable to find the Java method specified.

Script Commands:

All

Causes:

A problem occurred with the JNI bridge.

Actions:

Verify the QALoad installation was successful and re-install if not.

External Sources:

None

OFS_ERROR_00101

Unable to connect to server.

Description:

Unable to open a connection to the server.

Script Commands:

ofsHTTPDisconnect
ofsHTTPConnectToFormsServlet
ofsHTTPConnectToListenerServlet
ofsHTTPInitialFormsConnect
ofsSendRecv
ofsConnectToSocket

Causes:

The server may have a problem

Actions:

- Check the URL and its parameters.
- Ensure server is working properly.

External Sources:

None

OFS_ERROR_00102

Unable to communicate with server.

Description:

Unable to communicate with server on existing connection.

Script Commands:

ofsSendRecv
ofsSetServletMode
ofsConnectToSocket
ofsHTTPConnectToFormsServlet
ofsHTTPConnectToListenerServlet
ofsHTTPInitialFormsConnect

Causes:

If the URL is valid, the server is not accepting new connections.

Actions:

Check the URL.

External Sources:

None

OFS_ERROR_00103

Unable to disconnect.

Description:

Disconnecting from the server failed.

Script Commands:

ofsSocketDisconnect

ofsHTTPDisconnect

ofsServerSideDisconnect

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is configured and working properly.

External Sources:

None

OFS_ERROR_00104

Failed to send a heartbeat message.

Description:

Sending heartbeat message to the server failed.

Script Commands:

ofsConnectToSocket

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00105

Connection terminated by server.

Description:

The server has closed the connection.

Script Commands:

ofsHTTPConnectToFormsServlet ofsHTTPInitialFormsConnect

ofsHTTPConnectToListenerServlet ofsSendRecv

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00106

Server did not return the encryption keys.

Description:

Server did not return the encryption keys for the first Post request.

Script Commands:

ofsHTTPInitialFormsConnect

Causes:

Forms Server is not accepting new connections.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00107

Server reply data is invalid.

Description:

Server reply data is invalid.

Script Commands:

ofsSendRecv

Causes:

- The connection to the server may have a problem.
- If the Java msg displayed is null, server has terminated this Forms session.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00108

Failed to get the content length of the POST request.

Description:

Unable to find the content length HTTP header for the POST request being processed.

Script Commands:

ofsSendRecv

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00109

Failed to get the reply content.

Description:

Failed to get the reply content.

Script Commands:

ofsHTTPConnectToFormsServlet

ofsHTTPConnectToListenerServlet

Causes:

The URL is invalid.

Actions:

Check the URL.

External Sources:

None

OFS_ERROR_00110

Invalid URL: {}

Description:

The URL in use is not a valid URL.

Script Commands:

ofsHTTPDisconnect
ofsHTTPInitialFormsConnect
ofsHTTPConnectToFormsServlet
ofsSendRecv
ofsHTTPConnectToListenerServlet

Causes:

The URL is malformed.

Actions:

Check the URL specified.

External Sources:

None

OFS_ERROR_00111

Failed to store data from the server reply.

Description:

Failed to store data from the server reply.

Script Commands:

ofsSendRecv

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00112

Failed because the server sent this error message: {}

Description:

Server sent error message having a error code prefix of ORA-, FRM-, or APP-

Script Commands:

ofsSendRecv

Causes:

The server encountered an error.

Actions:

Review the Oracle error message specified and determine what action should be taken.

External Sources:

None

OFS_ERROR_00113

Failed while reading the server reply.

Description:

Failed while reading the server reply.

Script Commands:

ofsHTTPConnectToFormsServlet

ofsSendRecv

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00114

SSL Handshake failed.

Description:

The SSL handshake failed.

Script Commands:

ofsHTTPSDoSSLHandshake

Causes:

The connection to the server may have a problem.

Actions:

- Ensure server is working properly.
- Check client side SSL credentials.

External Sources:

None

OFS_ERROR_00115

Failed to close socket.

Description:

Unable to close socket.

Script Commands:

ofsSocketDisconnect

ofsHTTPDisconnect

Causes:

The connection to the server may have a problem.

Actions:

Ensure server is working properly.

External Sources:

None

OFS_ERROR_00116

Failed while processing server detail message. Unknown control handle.

Description:

An internal error occurred while attempting to process an unknown control handle.

Script Commands:

ofsSendRecv

Causes:

An unknown control handle was received from the server.

Actions:

External Sources:

OFS_ERROR_00117

ICX Ticket not found in OracleAppsLogin.

Description:

Unable to find ICX ticket.

Script Commands:

OracleAppsLogin

Causes:

The icx_ticket was not found in OracleAppsLogin.

Actions:

Check URL, user id, and password.

External Sources:

None

OFS_ERROR_00119

Failed to create the replay capture file.

Description:

OFS middleware was unable to open the replay capture file.

Script Commands:

ofsSetRunOptions

Causes:

- The user may not have write permission to LogFiles folder.
- The system may have run out of file descriptors.

Actions:

- Ensure that the user has write permission to LogFiles folder.
- Limit the number of VUs that are attempting to open replay capture files.

External Sources:

None

OFS_ERROR_00120

Unable to write to replay capture file.

Description:

Unable to write to the replay capture file.

Script Commands:

ofsHTTPConnectToFormsServlet ofsHTTPInitialFormsConnect

ofsHTTPConnectToListenerServlet ofsSendRecv

Causes:

Unable to write to replay capture file.

Actions:

Ensure that proper permissions for the file and log files exist.

External Sources:

None

OFS_ERROR_00121

JVM memory issues.

Description:

JVM memory issues.

Script Commands:

ofsSendRecv ofsHTTPConnectToListenerServlet

ofsHTTPConnectToFormsServlet ofsHTTPInitialFormsConnect

Causes:

The JVM may be out of memory.

Actions:

See additional information for more details.

External Sources:

None

OFS_ERROR_00122

Invalid argument: {}

Description:

An argument specified in the QALoad script command is not valid.

Script Commands:

ofsSendRecv ofsTabControlTopPage

ofsScroll ofsLOVSelection

ofsScrollSize ofsSetCursorPosition

ofsSetValue ofsSetRunOptions

Causes:

The argument specified is not valid for the script command.

Actions:

Check the arguments to the script command that failed.

External Sources:

None

OFS_ERROR_00123

Internal error

Description:

An error has occurred that was caught.

Script Commands:

All

Causes:

An error occurred.

Actions:

see additional information for more details.

External Sources:

None

SAP

SAP Playback Error Codes

QALoad displays error codes during playback for specific exception messages. While debugging, refer to the table below for error codes and descriptions that apply to SAPscripts.

Error code	Description
SAP_ERROR_00001	SAPGui: Failure connecting to <SAP server name>
SAP_ERROR_00002	SAP: CheckScreen match failure! Expected: <OK code> - <screen Name> - <Title>

	Returned: <OK code> - <screen Name> - <Title>
SAP_ERROR_00003	Control not found: <control name> (<Occurrence of the control>)
SAP_ERROR_00004	GetEvent returned an error
SAP_ERROR_00005	Handle to the GUILIB is NULL
SAP_ERROR_00006	Unable to load guilibx.dll
SAP_ERROR_00007	Unable to find function address
SAP_ERROR_00008	Null or required parameter missing to SAPLogin
SAP_ERROR_00009	Unable to connect to SAP Server: <hostname>, <system Number>
SAP_ERROR_00010	SAPLogin failed
SAP_ERROR_00011	'EVT_DYNPRO_INFO' not set in event structure
SAP_ERROR_00012	CheckScreen match failure!
SAP_ERROR_00013	Invalid parameter to SAPGetControlValue
SAP_ERROR_00014	Control not found in SPGetControlValue: <control name> (<Occurrence of the control>)
SAP_ERROR_00015	Unable to find PFkey: <key name>
SAP_ERROR_00016	Invalid parameter passed
SAP_ERROR_00017	Value must be 'TRUE' or 'FALSE' for CheckBox or RadioButton
SAP_ERROR_00018	Menu sub-string <current menu> in <menu structure> not found
SAP_WARNING_00019	Statusbar = <status bar value>

SAP_ERROR_00001

The SAPGUI client cannot connect to the specified server.

Description:

The SAPGUI client did not return an event indicating that the connection to the SAP server was successful.

Script Commands:

SAPGuiConnect

Causes:

- The SAP server is not accepting connections.
- The network latency is too great due to excessive traffic.
- The client machine is low on resources.

Actions:

- Ensure the SAP server is available and able to accept connections.
- Ensure the network is not overloaded with other traffic.
- Ensure the client machine has adequate resources.

External Sources:

None

SAP_ERROR_00002

The SAPGUI client found that a different window was displayed than during the record session.

Description:

The SAPGUI client found that the active window displayed at replay did not match the screen name, title, and OK code captured during the record session.

Script Commands:

SAPGuiCheckScreen

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00003

The control referenced by the API command cannot be found.

Description:

The SAP client did not find the control among the active control list when trying to set the cursor or control value.

Script Commands:

Do_SAPSetCursor

Do_SAPSetCtrlValue

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00004

The SAP GetEvent call returned an error message.

Description:

The send did not succeed. The SAP GetEvent call, which returns a confirmation that the send was successful, returned a failure message.

Script Commands:

Do_SAPLogin Do_SAPSendOKCode

Do_SAPSendPFKey Do_SAPSendReturn

Do_SAPSendEvent Do_SAPSendDblClick

Do_SAPSendMenu

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00005

The GUILIB handle is NULL. The SAP library is not loaded.

Description:

The player did not load the SAP GUI DLL correctly.

Script Commands:

Do_SAPLogoff Do_SAPSendOKCode

Do_SAPSendPFKey Do_SAPSendReturn

Do_SAPSendEvent Do_SAPSendDblClick

Do_SAPSendMenu

Causes:

- The SAP client may not be installed correctly.
- QALoad may not be installed correctly.

- The replay machine may be low on system resources.

Actions:

- Ensure that the SAP client and QALoad are installed correctly.
- Ensure that the replay machine has sufficient resources to replay the scripts.

External Sources:

None

SAP_ERROR_00006

The player is unable to load the SAP client library DLL, guilibx.dll

Description:

The call to the LoadLibrary function to load the sapguix.dll failed. The entry point for the function could not be found.

Script Commands:

Do_SAPInit

Causes:

- The SAP client may not be installed correctly.
- QALoad may not be installed correctly.
- The replay machine may be low on system resources.

Actions:

- Ensure that the SAP client and QALoad are installed correctly.
- Ensure that the replay machine has sufficient resources to replay the scripts.

External Sources:

None

SAP_ERROR_00007

The player cannot find the function in the guilibx.dll.

Description:

The call to the GetProcAddress function to find the entry point for the specified function failed.

Script Commands:

Do_SAPInit

Causes:

- The SAP client may not be installed correctly.
- QALoad may not be installed correctly.
- The replay machine may be low on system resources.

Actions:

- Ensure that the SAP client and QALoad are installed correctly.
- Ensure that the replay machine has sufficient resources to replay the scripts.

External Sources:

None

SAP_ERROR_00008

A required parameter for the SAPLogin call is not provided.

Description:

One of the required parameters for the DO_SAPLogin call (hostname, systemNo, username, and password) was not provided.

Script Commands:

Do_SAPLogin

Causes:

- The login parameters provided to the DO_SAPLogin call are incorrect.
- One of the required parameters is missing.

Actions:

Review the parameters for the DO_SAPLogin call and ensure all required parameters are provided and are correct.

External Sources:

None

SAP_ERROR_00009

Unable to connect to the SAP Server.

Description:

The DO_SAPLogin call could not connect to the SAP server specified by the hostname and systemNo parameters.

Script Commands:

Do_SAPLogin

Causes:

- The hostname or systemNo parameters provided to the DO_SAPLogin call are incorrect.
- The SAP server specified by the DO_SAPLogin parameters is not online and able to receive connections.
- The network may be congested with excessive traffic.

Actions:

- Review the parameters for the DO_SAPLogin call and ensure all required parameters are provided and are correct.
- Ensure the SAP server is online and able to receive connections.
- Ensure the network is not overburdened with traffic.

External Sources:

SAP_ERROR_00010

The SAPLogin call returned with an error.

Description:

The DO_SAPLogin call did not succeed in logging into the SAP server.

Script Commands:

Do_SAPLogin

Causes:

The login credentials may be incorrect.

Actions:

Ensure the parameters provided to the DO_SAPLogin call are correct.

External Sources:

None

SAP_ERROR_00011

The SAP event structure does not contain the 'EVT_DYNPRO_INFO' setting.

Description:

The SAP event structure does not have the EVT_DNYPRO_INFO tag set, indicating that SAP did not receive an event for the CheckScreen action.

Script Commands:

Do_SAPCheckScreen

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00012

The SAP client found a that different window was displayed than during the record session.

Description:

The SAP client found that the active window displayed at replay did not match the screen name, title, and OK code captured during the record session.

Script Commands:

Do_SAPCheckScreen

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00013

A parameter passed to SAPGetControlValue is invalid.

Description:

Either the ctrlname or buffer parameter is NULL or the ctrlOcc parameter is less than zero.

Script Commands:

Do_SAPGetControlValue

Causes:

- The record values were altered or an incorrect or NULL value in the script.
- The capture did not convert properly. Ensure that the convert values match the capture values.

Actions:

Ensure the parameters to SAPGetControlValue are correct.

External Source:

None

SAP_ERROR_00014

The control referenced by the SAPGetControlValue command cannot be found.

Description:

The SAP client did not find the control among the active control list when trying to get the control value.

Script Commands:

Do_SAPGetControlValue

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script that follows the current server responses to events in the application

External Sources:

None

SAP_ERROR_00015

The DO_SAPSendPFKey was unable to send the specified function key.

Description:

The function did not find the function key among the enumerated function keys supplied by the keyboard locale.

Script Commands:

Do_SAPSendPFKey

Causes:

The replay machine keyboard locale is different from the locale on the machine when the session was recorded.

Actions:

- Modify the script to accommodate the new server behavior.
- Record a new script for the keyboard locale that exists on the playback machine.

External Sources:

None

SAP_ERROR_00016

A parameter passed to the function is invalid.

Description:

A parameter passed to this function is incorrect. Use the Function Wizard as an aid in determining the correct values.

Script Commands:

Do_SAPSetCtrlValue

Do_SAPSetCursor

Causes:

- The record values were altered or an incorrect or NULL value was placed in the script.
- The capture did not convert properly. Ensure that the convert values match the capture values.

Actions:

Ensure the parameters passed to the function are correct.

External Sources:

None

SAP_ERROR_000017

The value passed to SAPSetControl must be "TRUE' or 'FALSE'

Description:

The only valid values that can be set for Checkbox or RadioButton controls are TRUE and FALSE. The buffer parameter for these controls must be set to either 'TRUE' or 'FALSE'.

Script Commands:

Do_SAPSetCtrlValue

Causes:

- The record values were altered or an incorrect or NULL value was placed in the script.
- The capture did not convert properly. Ensure that the convert values match the capture values.

Actions:

Ensure the parameters passed to the function are correct.

External Sources:

None

SAP_ERROR_00018

The menu item indicated by the menu and substring parameters is not in the current SAP menu.

Description:

The current menu structure does not contain either the current menu or the menu substring indicated as the menustring parameter to the DO_SAPSendMenu call.

Script Commands:

Do_SAPSendMenu

Causes:

- The record values were altered or an incorrect or NULL value was placed in the script as the parameter for the DO_SAPSendMenu call.
- The capture did not convert properly. Ensure that the convert values match the capture values.

Actions:

Ensure the parameters passed to the function are correct.

External Sources:

None

SAP_WARNING_00019

An unexpected value is displayed in the status bar.

Description:

The SAPGUICheckStatusBar function returned a different value for the status bar contents than the one captured during the record session.

Script Commands:

Causes:

The SAP server configuration changed so that there is unexpected script behavior.

Actions:

No actions are required.

Note: This may show that there have been changes to the SAP environment since the recorded session that may affect replay.

External Sources:

None

Winsock

Winsock Playback Error Codes

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

Error code	Description
WSK_ABORT_00001	ERROR with ACE_Thread::getspecific, TLS is NULL!
WSK_ABORT_00002	VUser Crashed in the WSK Middleware. VUser id: <task_id>
WSK_ERROR_00003	Connection handle greater than allowed maximum. Connection handle: <Connect Handle>, allowed maximum open connection: <maximum value>
WSK_ERROR_00004	Connection handle appears not to be in use. Connection handle: <connection handle>
WSK_ERROR_00005	Connection handle appears to already be in use. Connection handle: <connection handle>
WSK_ABORT_00006	Out of memory!
WSK_ERROR_00007	Unable to create socket. Error: <socket error number>
WSK_ERROR_00008	<Address> is not a legitimate Internet address
WSK_ERROR_00009	Unable to connect. Error: <socket error number>
WSK_ABORT_00010	Input buffer set to NULL
WSK_ABORT_00011	Invalid value in input buffer
WSK_ABORT_00012	Invalid number of characters in buffer
WSK_ERROR_00013	Failed to call send function. Error: <socket error number>
WSK_ERROR_00014	Failed to call sendto function. Error: <socket error number>
WSK_ERROR_00015	Failed to call recv function. Error: <socket error number>
WSK_ERROR_00016	Failed to call recvfrom function. Error: <socket error number>
WSK_ERROR_00017	Failed to call closesocket/close function. Error: <socket error number>
WSK_ERROR_00018	Failed to call setsockopt function. Error: <socket error number>
WSK_ERROR_00019	Failed to call getsockname function. Error: <socket error number>

WSK_ERROR_00020	Failed to call bind function. Error: <socket error number>
WSK_ERROR_00021	Failed to call listen function. Error: <socket error number>
WSK_ERROR_00022	Failed to call accept function. Error: <socket error number>
WSK_ERROR_00023	Failed to call select function. Error: <socket error number>
WSK_ERROR_00024	Failed to call ioctlsocket/ioctl function. Error: <socket error number>
WSK_WARNING_00025	Failed to call shutdown function. Error: <socket error number>

WSK_ABORT_00001

ERROR with ACE_Thread::getspecific, TLS is NULL!

Description:

Serious error. Thread local storage (TLS) is NULL.

Script Commands:

All WSK commands

Causes:

- When allocating thread local memory, insufficient memory exists to associate the value with the key.
- ACE Player error.

Actions:

Severe error. Call Technical Support immediately.

External Sources:

<ACE library>

ACE_Thread::getspecific(ACE)thread_key_t key, void **valuep)

Stores the current value bound to <key> for the calling thread into the location pointed to by <valuep>.

WSK_ABORT_00002

VUser crashed in the WSK Middleware.

Description:

Virtual user catches exception, which is in the Winsock (WSK) middleware.

Script Commands:

All Winsock commands

Causes:

When C++ exceptions occur in Winsock script, or Winsock script crashes when run in player.

Actions:

Check Winsock log file to determine which command causes this.

External Sources:

None

WSK_ERROR_00003

Connection handle greater than allowed maximum.

Description:

Internal function: CheckValidHandle() generates this error message. Each time a call is made for Sockets library functions, the connection handle is checked. The default Maximum open connection is set at 33.

Script Commands:

DO_WSK_Socket	DO_WSK_GetSocket	DO_WSK_ExpectAny
DO_WSK_Send	DO_WSK_IsWriteable	DO_WSK_Quiet
DO_WSK_Sendto	DO_WSK_Shutdown	DO_WSK_Setsockopt
DO_WSK_Read	GetRemotePort	DO_WSK_Bind
DO_WSK_Recvfrom	GetRemoteAddr	DO_WSK_Accept
DO_WSK_ExpectExpr	DO_WSK_Connect	DO_WSK_IsReadable
DO_WSK_ExpectAnyExpr	DO_WSK_SendAll	DO_WSK_IoctlSocket
DO_WSK_Closesocket	DO_WSK_Write	GetLocalPort
DO_WSK_Getsockname	DO_WSK_Recv	GetLocalAddr
DO_WSK_Listen	DO_WSK_Expect	

Causes:

- Open too many connects.
- Forget to close connect when done.

Actions:

Check the Winsock log file to determine which command causes this.

External Sources:

None

WSK_ERROR_00004

Connection handle appears not to be in use.

Description

Internal function: CheckValidHandle() generates this error message. Each time a call is made for Sockets library functions, the opened connection handle is checked to see if socket is valid. If not, this error message is returned.

Script Commands:

DO_WSK_Connect DO_WSK_Send DO_WSK_SendAll

DO_WSK_Sendto DO_WSK_Write DO_WSK_Read

DO_WSK_Recv DO_WSK_Recvfrom DO_WSK_Expect

DO_WSK_ExpectAny DO_WSK_ExpectExpr DO_WSK_ExpectAnyExpr

DO_WSK_Quiet DO_WSK_Closesocket DO_WSK_Setsockopt

DO_WSK_Getsockname DO_WSK_Bind DO_WSK_Listen

DO_WSK_Accept DO_WSK_GetSocket DO_WSK_IsReadable

DO_WSK_IsWriteable DO_WSK_IoctlSocket DO_WSK_Shutdown

GetLocalPort GetRemotePort GetLocalAddr

GetRemoteAddr

Causes

Function socket doesn't successfully create a connection handle. When other sockets library functions use this connection handle, this error message occurs.

Actions:

Check Winsock log file to determine which command causes this.

External Sources:

None

WSK_ERROR_00005

Connection handle already appears to be in use.

Description:

Internal function: CheckValidHandle() generates this error message. Each time before a call socket() function is made, CheckValidHandle() checks to see if the connection handle is already used.

Script Commands:

DO_WSK_Socket

Causes:

Connection handle is already used. Either forget to close it or input wrong connection handle.

Actions:

Check Winsock log file to get connection handle. Change the first parameter in DO_WSK_Socket to try another connection handle in Winsock script.

External Sources:

None

WSK_ABORT_00006

Out of memory.

Description:

There is insufficient memory available when allocating memory blocks.

Script Commands:

EscapeStr
DO_WSK_Send
DO_WSK_Write
DO_WSK_SendAll
DO_WSK_Expect
DO_WSK_Sendto
DO_WSK_ExpectExpr
DO_WSK_Accept
DO_WSK_ExpectAny
ScanExpr
DO_WSK_ExpectAnyExpr
SkipExpr
DO_WSK_Quiet
DO_WSK_Socket
DO_WSK_Read

Causes:

There is insufficient memory available.

Actions:

Deallocate or free previously allocated memory block when not in use.

External Sources:

None

WSK_ERROR_00007

Unable to create socket.

Description:

Fail to call socket function. Socket function creates a socket that is bound to a specific service provider.

Script Commands:

DO_WSK_Socket

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.

WSAENETDOWN (10050)	The network subsystem or the associated service provider has failed.
WSAEAFNOSUPPORT (10047)	The specified address family is not supported.
WSAEINPROGRESS (10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEMFILE (10024)	No more socket descriptors are available.
WSAENOBUFS (10055)	No buffer space is available. The socket cannot be created.
WSAEPROTONOSUPPORT (10043)	The specified protocol is not supported.
WSAEPROTOTYPE (10041)	The specified protocol is the wrong type for this socket.
WSAESOCKTNOSUPPORT (10044)	The specified socket type is not supported in this address family.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EACCES(13)	Permission to create a socket of the specified type or protocol is denied.
EMFILE (24)	The per-process descriptor table is full.
ENOMEM(12)	Insufficient user memory is available.
ENOSR(63)	There were insufficient STREAMS resources available to complete the operation.
EPROTONOSUPPORT(120)	The protocol type or the specified protocol is not supported within this domain.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EPROTONOSUPPORT (93)	The protocol type or the specified protocol is not supported within this domain.
EAFNOSUPPORT(97)	The implementation does not support the specified address family.
ENFILE (23)	Not enough kernel memory to allocate a new socket structure.
EMFILE (24)	Process file table overflow.
EACCES (13)	Permission to create a socket of the specified type or protocol is denied.

ENOBUFS(105) or ENOMEM (12)	Insufficient memory is available. The socket cannot be created until sufficient resources are freed.
EINVAL(22)	Unknown protocol, or protocol family not available.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00008

Input internet address is not a legitimate internet address.

Description:

QALoad uses function inet_addr() to convert a string containing the Internet Protocol dotted address into a proper address. If the string does not contain a legitimate Internet address, QALoad returns this error message.

Script Commands:

DO_WSK_Bind

DO_WSK_Connect

DO_WSK_Sendto

Causes:

The string does not contain an Ipv4 legitimate Internet address. For example, if a portion of an "a.b.c.d" address exceeds 255.

Actions:

Check Winsock log file for errors.

External sources:

Copy from MSDN: Values specified using the "." notation take one of the following forms: a.b.c.d a.b.c a.b a. When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the 4 bytes of an Internet address. When an Internet address is viewed as a 32-bit integer quantity on the Intel architecture, the bytes referred to above appear as "d.c.b.a". That is, the bytes on an Intel processor are ordered from right to left.

The parts that make up an address in "." notation can be decimal, octal, or hexadecimal as specified in the C language. Numbers that start with "0x" or "0X" imply hexadecimal. Numbers that start with "0" imply octal. All other numbers are interpreted as decimal.

Internet address value meaning: "4.3.2.16" Decimal, "004.003.002.020" Octal, "0x4.0x3.0x2.0x10" Hexadecimal, "4.003.002.0x10" mix.

Note: The following notations are only used by Berkeley, and nowhere else on the Internet. For compatibility with their software, they are supported as specified. When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right-most 2 bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as "128.net.host". When a two-part address is specified, the last part is interpreted as a 24-bit quantity and placed in the right-most 3 bytes of the network address. This makes the two-part address format convenient for specifying Class

A network addresses as "net.host". When only one part is given, the value is stored directly in the network address without any byte rearrangement.

WSK_ERROR_00009

Unable to connect.

Description:

Fail to call connect function. The connect function establishes a connection to a specified socket.

Script Commands:

DO_WSK_Connect

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEADDRINUSE (10048)	The socket's local address is already in use and the socket was not marked to allow address reuse with SO_REUSEADDR. This error usually occurs when executing bind. It could be delayed until this function if the bind was to a partially wildcard address (involving ADDR_ANY) and if a specific address needs to be committed at the time of this function.
WSAEINTR (10004)	A blocking Windows Sockets 1.1 call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS (10036)	A blocking Windows Sockets 1.1 call is in progress or the service provider is still processing a callback function.
WSAEALREADY (10037)	A nonblocking connect call is in progress on the specified socket. Note that in order to preserve backward compatibility, this error is reported as WSAEINVAL to Windows Sockets 1.1 applications that link to either Winsock.dll or Wsock32.dll.
WSAEADDRNOTAVAIL (10049)	The remote address is not a valid address (such as ADDR_ANY).
WSAEAFNOSUPPORT (10047)	Addresses in the specified family cannot be used with this socket.
WSAECONNREFUSED (10061)	The attempt to connect was forcefully rejected.
WSAEFAULT(10014)	The name or the namelen parameter is not a valid part of the user address space; the namelen parameter is too small; or the name parameter contains incorrect address format for the associated address

	family.
WSAEINVAL (10022)	The parameter s is a listening socket.
WSAEISCONN (1056)	The socket is already connected (connection-oriented sockets only).
WSAENETUNREACH (10051)	The network cannot be reached from this host at this time.
WSAENOBUFS	No buffer space is available. The socket cannot be connected.
WSAENOTSOCK (1055)	The descriptor is not a socket.
WSAETIMEDOUT (1060)	Attempt to connect timed out without establishing a connection.
WSAEWOULDBLOCK (10035)	The socket is marked as non-blocking and the connection cannot be completed immediately.
WSAEACCES (10013)	Attempt to connect datagram socket to broadcast address failed because setsockopt option SO_BROADCAST is not enabled.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EACCES(13)	Search permission is denied for a component of the path prefix of the pathname in name.
EADDRINUSE(125)	The address is already in use.
EADDRNOTAVAIL(126)	The specified address is not available on the remote machine.
EAFNOSUPPORT(124)	Addresses in the specified address family cannot be used with this socket.
EALREADY(149)	The socket is non-blocking and a previous connection attempt has not yet been completed.
EBADF(9)	This is not a valid descriptor.
ECONNREFUSED(146)	The attempt to connect was forcefully rejected. The calling program should close(2) the socket descriptor, and issue another socket(3N) call to obtain a new descriptor before attempting another connect() call.
EINPROGRESS(150)	The socket is non-blocking and the connection cannot be completed immediately. It is possible to select(3C) for completion by selecting the socket for writing. However, this is only possible if the socket STREAMS module is the topmost module on the protocol stack with a write service procedure. This is normally case.
EINTR(4)	The connection attempt was interrupted before any data arrived by the delivery of a signal.

EINVAL(22)	The namelen parameter is not the size of a valid address for the specified address family.
EIO(5)	An I/O error occurred while reading from or writing to the file system.
EISCONN(133)	The socket is already connected.
ELOOP(90)	Too many symbolic links were encountered in translating the pathname in name.
ENETUNREACH(128)	The network is not reachable from this host.
ENOENT(2)	A component of the path prefix of the pathname in name does not exist.
ENOENT(2)	The socket referred to by the pathname in name does not exist.
ENOSR(63)	There were insufficient STREAMS resources available to complete the operation.
ENXIO(6)	The server exited before the connection was complete.
ETIMEDOUT(145)	Connection establishment timed out without establishing a connection.
EWOULDBLOCK(11)	The socket is marked as non-blocking, and the requested operation would block.

Unix

The following errors are specific to connecting names in the UNIX domain.

Note: These errors may not apply in future versions of the UNIX IPC domain.

Error Code	Description
ENOTDIR(20)	A component of the path prefix of the pathname in name is not a directory.
ENOTSOCK(95)	Name is not a socket.
EPROTOTYPE(98)	The file referred to by name is a socket of a type other than type s (for example, s is a SOCK_DGRAM socket, while name refers to a SOCK_STREAM socket).

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	The file descriptor is not a valid index in the descriptor table.
EFAULT(14)	The socket structure address is outside the user's address space.
ENOTSOCK(88)	The file descriptor is not associated with a socket.

EISCONN(106)	The socket is already connected.
ECONNREFUSED(111)	No one listening on the remote address.
ETIMEDOUT(110)	Timeout while attempting connection. The server may be too busy to accept new connections. Note that for IP sockets, the timeout may be very long when syncookies are enabled on the server.
ENETUNREACH(101)	Network is unreachable.
EADDRINUSE(98)	Local address is already in use.
EINPROGRESS(115)	The socket is non-blocking and the connection cannot be completed immediately. It is possible to select(2) or poll(2) for completion by selecting the socket for writing. After select indicates writability, use getsockopt(2) to read the SO_ERROR option at level SOL_SOCKET. This indicates whether connect completed successfully (SO_ERROR is zero) or unsuccessfully. (SO_ERROR is one of the usual error codes listed here, explaining the reason for the failure).
EALREADY(114)	The socket is non-blocking and a previous connection attempt has not yet been completed.
EAGAIN(11)	No more free local ports or insufficient entries in the routing cache. For PF_INET, see the net.ipv4.ip_local_port_range sysctl in ip(7) on how to increase the number of local ports.
EAFNOSUPPORT(97)	The passed address didn't have the correct address family in its sa_family field.
EACCES(13), EPERM(1)	The user tried to connect to a broadcast address without having the socket broadcast flag enabled, or the connection request failed because of a local firewall rule.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ABORT_00010

Input buffer set to NULL.

Description:

A pointer to a buffer to be converted by function DO_WSK_HexDecode is NULL.

Script Commands:

DO_WSK_HexDecode

Causes:

Improperly handle char type pointer.

Actions:

Check Winsock log file for errors.

External Sources:

None

WSK_ABORT_00011

Invalid value in input buffer.

Description:

Occurs when DO_WSK_HexDecode is called. DO_WSK_HexDecode converts hexadecimal characters to binary data suitable for sending to a connection using DO_WSK_Write(). If input string is not hexadecimal type data, this error occurs.

Script Commands:

DO_WSK_HexDecode

Causes:

Wrong input string.

Actions:

Check Winsock log file for errors.

External Sources:

None

WSK_ABORT_00012

Invalid number of characters in buffer.

Description:

Occurs when DO_WSK_HexDecode is called. DO_WSK_HexDecode converts hexadecimal characters to binary data suitable for sending to a connection using DO_WSK_Write(). If the number of input bytes is not even, this error occurs.

Script Commands:

DO_WSK_HexDecode

Causes:

Wrong input string.

Actions:

Check Winsock log file for errors.

External Sources:

None

WSK_ERROR_000013

Error during send.

Description:

Fail to call send function. The send function sends data on a connected socket.

Script Commands:

DO_WSK_Send()

DO_WSK_SendAll()

DO_WSK_Write()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEACCES (10013)	The requested address is a broadcast address, but the appropriate flag was not set. Call setsockopt with the SO_BROADCAST socket option to enable use of the broadcast address.
WSAEINTR (10004)	A blocking Windows Sockets 1.1 call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS (10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEFAULT (10014)	The buf parameter is not completely contained in a valid part of the user address space.
WSAENETRESET (10052)	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
WSAENOBUFS (10055)	No buffer space is available.
WSAENOTCONN (10057)	The socket is not connected.
WSAENOTSOCK (10038)	The descriptor is not a socket.
WSAEOPNOTSUPP (10045)	MSG_OOB was specified, but the socket is not stream-style such as type SOCK_STREAM; OOB data is not supported in the communication domain associated with this socket; or the socket is unidirectional and supports only receive operations.
WSAESHUTDOWN (10058)	The socket has been shut down; it is not possible to send on a socket after shutdown has been invoked with how set to SD_SEND or SD_BOTH.
WSAEWOULDBLOCK (10035)	The socket is marked as non-blocking and the requested operation

	would block.
WSAEMSGSIZE (10040)	The socket is message oriented, and the message is larger than the maximum supported by the underlying transport.
WSAEHOSTUNREACH (10065)	The remote host cannot be reached from this host at this time.
WSAEINVAL (10022)	The socket has not been bound with bind; or an unknown flag was specified; or MSG_OOB was specified for a socket with SO_OOBINLINE enabled.
WSAECONNABORTED (10053)	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
WSAECONNRESET (10054)	The virtual circuit was reset by the remote side executing a hard or abortive close. For UPD sockets, the remote host was unable to deliver a previously sent UDP datagram and responded with a "Port Unreachable" ICMP packet. The application should close the socket as it is no longer usable.
WSAETIMEDOUT (10060)	The connection has been dropped because of a network failure or because the system on the other end went down without notice.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	s is an invalid file descriptor.
EINTR(4)	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL(22	tolen is not the size of a valid address for the specified address family.
EMSGSIZE(97)	The socket requires that message be sent atomically and the message was too long.
ENOMEM(12)	There was insufficient memory available to complete the operation.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK(95)	Not a socket.
EWOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	An invalid descriptor was specified.
ENOTSOCK(88)	The argument s is not a socket.
EFAULT(14)	An invalid user space address was specified for a parameter.
EMSGSIZE(90)	The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.
EAGAIN (11) or EWOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.
ENOBUFS(105)	The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion. (Normally, this does not occur in Linux. Packets are just silently dropped when a device queue overflows.)
EINTR(4)	A signal occurred.
ENOMEM(12)	No memory available.
EINVAL(22)	Invalid argument passed.
EPIPE(32)	The local end has been shut down on a connection oriented socket. In this case, the process also receives a SIGPIPE unless MSG_NOSIGNAL is set.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000014

Fail to call sendto function.

Description:

The sendto function sends data to a specific destination.

Script Commands:

DO_WSK_Sendto()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEACCES (10013)	The requested address is a broadcast address, but the appropriate flag was not set. Call setsockopt with the SO_BROADCAST parameter to allow the use of the broadcast address.
WSAEINVAL (10022)	An unknown flag was specified, or MSG_OOB was specified for a socket with SO_OOBINLINE enabled.
WSAEINTR (10004)	A blocking Windows Sockets 1.1 call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS (10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEFAULT (10014)	The buf or to parameters are not part of the user address space, or the tolen parameter is too small.
WSAENETRESET (10052)	The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress.
WSAENOBUFS (10055)	No buffer space is available.
WSAENOTCONN (10057)	The socket is not connected (connection-oriented sockets only).
WSAENOTSOCK (10038)	The descriptor is not a socket.
WSAEOPNOTSUPP (10045)	MSG_OOB was specified, but the socket is not stream-style such as type SOCK_STREAM; OOB data is not supported in the communication domain associated with this socket; or the socket is unidirectional and supports only receive operations.
WSAESHUTDOWN (10058)	The socket has been shut down; it is not possible to sendto on a socket after shutdown has been invoked with how set to SD_SEND or SD_BOTH.
WSAEWOULDBLOCK (10035)	The socket is marked as non-blocking and the requested operation would block.
WSAEMSGSIZE (10040)	The socket is message oriented and the message is larger than the maximum supported by the underlying transport.
WSAEHOSTUNREACH (10065)	The remote host cannot be reached from this host at this time.
WSAECONNABORTED (10053)	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
WSAECONNRESET (10054)	The virtual circuit was reset by the remote side executing a hard or abortive close. For UPD sockets, the remote host was unable to deliver a previously sent UDP datagram and responded with a "Port

	Unreachable" ICMP packet. The application should close the socket as it is no longer usable.
WSAEADDRNOTAVAIL (10049)	The remote address is not a valid address, for example, ADDR_ANY.
WSAEAFNOSUPPORT (10047)	Addresses in the specified family cannot be used with this socket.
WSAEDESTADDRREQ (10039)	A destination address is required.
WSAENETUNREACH (10051)	The network cannot be reached from this host at this time.
WSAETIMEDOUT (10060)	The connection has been dropped because of a network failure or because the system on the other end went down without notice.
WOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.
EBADF(9)	An invalid descriptor was specified.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	s is an invalid file descriptor.
EINTR(4)	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EINVAL(22	tolen is not the size of a valid address for the specified address family.
EMSGSIZE(97)	The socket requires that message be sent atomically, and the message was too long.
ENOMEM(12)	There was insufficient memory available to complete the operation.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK(95)	s is not a socket.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
ENOTSOCK(88)	The argument s is not a socket.
EFAULT(14)	An invalid user space address was specified for a parameter.
EMSGSIZE(90)	The socket requires that message be sent atomically, and the size of the message to be sent made this impossible.

EAGAIN (11) or EWOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.
ENOBUFS(105)	The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion. (Normally, this does not occur in Linux. Packets are just silently dropped when a device queue overflows.)
EINTR(4)	A signal occurred.
ENOMEM(12)	No memory available.
EINVAL(22)	Invalid argument passed.
EPIPE(32)	The local end has been shut down on a connection oriented socket. In this case, the process also receives a SIGPIPE unless MSG_NOSIGNAL is set.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000015

Error during recv.

Description:

Fail to call recv function. The recv function receives data from a connected or bound socket.

Script Commands:

DO_WSK_Recv()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEFAULT (10014)	The buf parameter is not completely contained in a valid part of the user address space.
WSAENOTCONN?10057)	The socket is not connected.

WSAEINTR(10004)	The (blocking) call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENETRESET(10052)	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
WSAENOTSOCK(10038)	The descriptor is not a socket.
WSAEOPNOTSUPP(10045)	MSG_OOB was specified, but the socket is not stream-style such as type SOCK_STREAM, OOB data is not supported in the communication domain associated with this socket, or the socket is unidirectional and supports only send operations.
WSAESHUTDOWN(10058)	The socket has been shut down; it is not possible to receive on a socket after shutdown has been invoked with how set to SD_RECEIVE or SD_BOTH.
WSAEWOULDBLOCK(10035)	The socket is marked as non-blocking and the receive operation would block.
WSAEMSGSIZE(10040)	The message was too large to fit into the specified buffer and was truncated.
WSAEINVAL(10022)	The socket has not been bound with bind, or an unknown flag was specified, or MSG_OOB was specified for a socket with SO_OOBINLINE enabled, or (for byte stream sockets only) len was zero or negative.
WSAECONNABORTED(10053)	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
WSAETIMEDOUT(10060)	The connection has been dropped because of a network failure or because the peer system failed to respond.
WSAECONNRESET(10054)	The virtual circuit was reset by the remote side executing a hard or abortive close. The application should close the socket as it is no longer usable. On a UPD-datagram socket this error would indicate that a previous send operation resulted in an ICMP "Port Unreachable" message.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	s is an invalid file descriptor.
EINTR(4)	The operation was interrupted by delivery of a signal before any data was available to be received.
EIO(5)	An I/O error occurred while reading from or writing to the file system.

ENOMEM(12)	There was insufficient user memory available for the operation to complete.
ENOSR(63)	There were insufficient STREAMS resources avail- able for the operation to complete.
ENOTSOCK(95)	s is not a socket.
ESTALE(151)	A stale NFS file handle exists.
EWOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	The argument s is an invalid descriptor.
ECONNREFUSED(111)	A remote host refused to allow the network connection, typically because it is not running the requested service.
ENOTCONN(107)	The socket is associated with a connection-oriented protocol and has not been connected (see connect(2) and accept(2)).
ENOTSOCK(88)	The argument s does not refer to a socket.
EAGAIN(11)	The socket is marked non-blocking and the receive operation would block, or a receive timeout had been set and the timeout expired before data was received.
EINTR(4)	The receive was interrupted by delivery of a signal before any data were available.
EFAULT(14)	The receive buffer pointer(s) point outside the process's address space.
EINVAL(22)	Invalid argument passed.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000016

Fail to call recvfrom function.

Description:

Fail to call recvfrom function. The recvfrom function receives a datagram and stores the source address.

Script Commands:

DO_WSK_Recvfrom()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEFAULT(10014)	The buf or from parameters are not part of the user address space, or the fromlen parameter is too small to accommodate the peer address.
WSAEINTR(10004)	The (blocking) call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL(10022)	The socket has not been bound with bind, or an unknown flag was specified, or MSG_OOB was specified for a socket with SO_OOBINLINE enabled, or (for byte stream-style sockets only) len was zero or negative.
WSAEISCONN(10056)	The socket is connected. This function is not permitted with a connected socket, whether the socket is connection oriented or connectionless.
WSAENETRESET(10052)	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
WSAENOTSOCK(10038)	The descriptor is not a socket.
WSAEOPNOTSUPP(10045)	MSG_OOB was specified, but the socket is not stream-style such as type SOCK_STREAM, OOB data is not supported in the communication domain associated with this socket, or the socket is unidirectional and supports only send operations.
WSAESHUTDOWN(10058)	The socket has been shut down; it is not possible to recvfrom on a socket after shutdown has been invoked with how set to SD_RECEIVE or SD_BOTH.
WSAEWOULDBLOCK(10035)	The socket is marked as nonblocking and the recvfrom operation would block.
WSAEMSGSIZE (10040)	The message was too large to fit into the specified buffer and was truncated.

WSAETIMEDOUT(10060)	The connection has been dropped because of a network failure or because the system on the other end went down without notice.
WSAECONNRESET(10054)	The virtual circuit was reset by the remote side executing a hard or abortive close. The application should close the socket; it is no longer usable. On a UDP-datagram socket, this error indicates a previous send operation resulted in an ICMP Port Unreachable message.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	s is an invalid file descriptor.
EINTR(4)	The operation was interrupted by delivery of a signal before any data was available to be received.
EIO(5)	An I/O error occurred while reading from or writing to the file system.
ENOMEM(12)	There was insufficient user memory available for the operation to complete.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK(95)	s is not a socket.
ESTALE(151)	A stale NFS file handle exists.
EWOULDBLOCK(11)	The socket is marked non-blocking and the requested operation would block.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	The argument s is an invalid descriptor.
ECONNREFUSED(111)	A remote host refused to allow the network connection, typically because it is not running the requested service.
ENOTCONN(107)	The socket is associated with a connection-oriented protocol and has not been connected (see connect(2) and accept(2)).
ENOTSOCK(88)	The argument s does not refer to a socket.
EAGAIN(11)	The socket is marked non-blocking and the receive operation would block, or a receive timeout had been set and the timeout expired before data was received.

EINTR(4)	The receive was interrupted by delivery of a signal before any data were available.
EFAULT(14)	The receive buffer pointer(s) point outside the process's address space.
EINVAL(22)	Invalid argument passed.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000017

Failed to call closesocket/close function.

Description:

Fail to call closesocket function (close function on Solaris/Linux platform). This function closes an existing socket.

Script Commands:

DO_WSK_Closesocket

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.
WSAEINTR(10004)	The (blocking) Windows Socket 1.1 call was canceled through WSACancelBlockingCall.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENOTSOCK(10038)	The descriptor is not a socket.
WSAEWOULDBLOCK(10035)	The socket is marked as non-blocking and SO_LINGER is set to a nonzero time-out value.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	The fildes argument is not a valid file descriptor.
EINTR(4)	The close() function was interrupted by a signal.
EIO(5)	An I/O error occurred while reading from or writing to the file system.
ENOLINK(67)	The fildes argument is on a remote machine and the link to that machine is no longer active.
ENOSPC(28)	There was no free space remaining on the device containing the file.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	fd isn't a valid open file descriptor.
EINTR (4)	The close() call was interrupted by a signal.
EIO (5)	An I/O error occurred.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000018

Fail to call setsockopt function.

Description:

Fail to call setsockopt function. The setsockopt function sets a socket option.

Script Commands:

DO_WSK_Setsockopt

DO_WSK_Socket

DO_WSK_Accept

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEFAULT(10014)	optval is not in a valid part of the process address space, or optlen parameter is too small.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL(10022)	level is not valid, or the information in optval is not valid.
WSAENETRESET(10052)	Connection has timed out when SO_KEEPALIVE is set.
WSAENOPROTOOPT(10042)	The option is unknown or unsupported for the specified provider or socket (see SO_GROUP_PRIORITY limitations).
WSAENOTCONN(10057)	Connection has been reset when SO_KEEPALIVE is set.
WSAENOTSOCK(10038)	The descriptor is not a socket.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	The argument s is not a valid file descriptor.
ENOMEM(12)	There was insufficient memory available for the operation to complete.
ENOPROTOOPT(99)	The option is unknown at the level indicated.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK(95)	The argument s is not a socket.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	The argument s is not a valid descriptor.
ENOTSOCK(88)	The argument s is a file, not a socket.

ENOPROTOOPT(92)	The option is unknown at the level indicated.
EFAULT(14)	The address pointed to by optval is not in a valid part of the process address space. For getsockopt, this error may also be returned if optlen is not in a valid part of the process address space.
EINVAL(22)	optlen invalid in setsockopt.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_000019

Fail to call getsockname function.

Description:

Fail to call getsockname function. The setsockopt function sets a socket option.

Script Commands:

DO_WSK_Bind

DO_WSK_Getsockname

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this API.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEFAULT (10014)	The name or the namelen parameter is not a valid part of the user address space, or the namelen parameter is too small.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENOTSOCK (10038)	The descriptor is not a socket.
WSAEINVAL(10022)	The socket has not been bound to an address with bind, or ADDR_ANY is specified in bind but connection has not yet occurred.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	The argument s is not a valid file descriptor.
ENOMEM(12)	There was insufficient memory available for the operation to complete.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTSOCK (95)	The argument s is not a socket.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	The argument s is not a valid descriptor.
ENOTSOCK(88)	The argument s is a file, not a socket.
ENOBUFS(105)	Insufficient resources were available in the system to perform the operation.
EFAULT (914)	The name parameter points to memory not in a valid part of the process address space.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00020

Fail to call bind function.

Description:

Fail to call bind function. The bind function associates a local address with a socket.

Script Commands:

DO_WSK_Bind()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEACCES(10013)	Attempt to connect datagram socket to broadcast address failed because setsockopt option SO_BROADCAST is not enabled.
WSAEADDRINUSE(10048)	A process on the computer is already bound to the same fully qualified address and the socket has not been marked to allow address reuse with SO_REUSEADDR. For example, the IP address and port are bound in the af_inet case. (See the SO_REUSEADDR socket option under setsockopt.)
WSAEADDRNOTAVAIL(10049)	The specified address is not a valid address for this computer.
WSAEFAULT(10014)	The name or namelen parameter is not a valid part of the user address space, the namelen parameter is too small, the name parameter contains an incorrect address format for the associated address family, or the first two bytes of the memory block specified by name does not match the address family associated with the socket descriptor s.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL(10022)	The socket is already bound to an address.
WSAENOBUFS(10055)	Not enough buffers available, too many connections.
WSAENOTSOCK(10038)	The descriptor is not a socket.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EACCES(13)	The requested address is protected and the current user has inadequate permission to access it.
EADDRINUSE(125)	The specified address is already in use.
EADDRNOTAVAIL (126)	The specified address is not available on the local machine.
EBADF(9)	s is not a valid descriptor.
EINVAL(22)	namelen is not the size of a valid address for the specified address family.

EINVAL(22)	The socket is already bound to an address.
ENOSR (63)	There were insufficient STREAMS resources for the operation to complete.
ENOTSOCK(95)	s is a descriptor for a file, not a socket.
EACCES(13)	Search permission is denied for a component of the path prefix of the pathname in name.
EIO (5)	An I/O error occurred while making the directory entry or allocating the inode.
EISDIR (21)	A null pathname was specified.
ELOOP (90)	Too many symbolic links were encountered in translating the pathname in name.
ENOENT (2)	A component of the path prefix of the pathname in name does not exist.
ENOTDIR (20)	A component of the path prefix of the pathname in name is not a directory.
EROFS (30)	The inode would reside on a read-only file system.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	sockfd is not a valid descriptor.
EINVAL(22)	The socket is already bound to an address. This may change in the future: see linux/unix/sock.c for details.
EACCES (13)	The address is protected and the user is not the supervisor.
ENOTSOCK (88)	Argument is a descriptor for a file, not a socket.

Unix (AF-UNIX)

The following errors are specific to UNIX domain (AF_UNIX) sockets:

Error Code	Description
EINVAL(22)	The addrlen is wrong, or the socket was not in the AF_UNIX family.
EROFS (30)	The socket inode would reside on a read-only file system.

EFAULT (14)	my_addr points outside the user's accessible address space.
ENAMETOOLONG (36)	my_addr is too long.
ENOENT(2)	The file does not exist.
ENOMEM (12)	Insufficient kernel memory was available.
ENOTDIR (20)	A component of the path prefix is not a directory.
EACCES (13)	Search permission is denied on a component of the path prefix.
ELOOP (40)	Too many symbolic links were encountered in resolving my_addr.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00021

Fail to call listen function.

Description:

Fail to call listen function. The listen function places a socket in a state in which it is listening for an incoming connection.

Script Commands:

DO_WSK_Listen()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEADDRINUSE(10048)	The socket's local address is already in use and the socket was not marked to allow address reuse with SO_REUSEADDR. This error usually occurs during execution of the bind function, but could be delayed until this function if the bind was to a partially wildcard address (involving ADDR_ANY) and if a specific address needs to be committed at the time of this function.

WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL(10022)	The socket has not been bound with bind.
WSAEISCONN(10056)	The socket is already connected.
WSAEMFILE(10024)	No more socket descriptors are available.
WSAENOBUFS(10055)	No buffer space is available.
WSAENOTSOCK(10038)	The descriptor is not a socket.
WSAEOPNOTSUPP(10045)	The referenced socket is not of a type that supports the listen operation.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	The argument s is not a valid file descriptor.
ENOTSOCK(95)	The argument s is not a socket.
EOPNOTSUPP(122)	The socket is not of a type that supports the operation listen().

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EADDRINUSE(40)	Another socket is already listening on the same port.
EBADF(9)	The argument s is not a valid descriptor.
ENOTSOCK(88)	The argument s is not a socket.
EOPNOTSUPP(95)	The socket is not of a type that supports the listen operation.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00022

Fail to call accept function.

Description:

Fail to call accept function. The accept function permits an incoming connection attempt on a socket.

Script Commands:

DO_WSK_Accept()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAECONNRESET(10054)	An incoming connection was indicated, but was subsequently terminated by the remote peer prior to accepting the call.
WSAEFAULT(10014)	The addrlen parameter is too small, or addr is not a valid part of the user address space.
WSAEINTR(10004)	A blocking Windows Sockets 1.1 call was canceled through WSACancelBlockingCall.
WSAEINVAL(10022)	The listen function was not invoked prior to accept.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEMFILE(10024)	The queue is nonempty upon entry to accept and there are no descriptors available.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAENOBUFS(10055)	No buffer space is available.
WSAENOTSOCK(10038)	The descriptor is not a socket.
WSAEOPNOTSUPP(10045)	The referenced socket is not of a type that supports the listen operation.
WSAEWOULDBLOCK(10035)	The socket is marked as nonblocking and no connections are present to be accepted.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description

EBADF(9)	The descriptor is invalid.
EINTR(4)	The accept attempt was interrupted by the delivery of a signal.
EMFILE(24)	The per-process descriptor table is full.
ENODEV(19)	The protocol family and type corresponding to s could not be found in the netconfig file.
ENOMEM(12)	There was insufficient user memory available to complete the operation.
ENOSR(63)	There were insufficient STREAMS resources available to complete the operation.
ENOTSOCK(95)	The descriptor does not reference a socket.
EOPNOTSUPP(122)	The referenced socket is not of type SOCK_STREAM.
EPROTO(71)	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized or the connection has already been released.
EWOULDBLOCK(11)	The socket is marked as non-blocking and no connections are present to be accepted.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EAGAIN(11) or EWOULDBLOCK(11)	The socket is marked non-blocking and no connections are present to be accepted.
EBADF(9)	The descriptor is invalid.
ENOTSOCK(88)	The descriptor references a file, not a socket.
EOPNOTSUPP(95)	The referenced socket is not of type SOCK_STREAM.
EINTR(4)	The system call was interrupted by a signal that was caught before a valid connection arrived.
ECONNABORTED(103)	A connection has been aborted.
EINVAL (22)	Socket is not listening for connections.
EMFILE (24)	The per-process limit of open file descriptors has been reached.
ENFILE (24)	The system maximum for file descriptors has been reached.
EFAULT (14)	The addr parameter is not in a writable part of the user address space.

ENOBUFS(105), ENOMEM(12)	Not enough free memory. This often means that the memory allocation is limited by the socket buffer limits, not by the system memory.
EPROTO(71)	Protocol error.
EPERM(1)	Firewall rules forbid connection.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00023

Fail to call select function.

Description:

Fail to call select function. The select function determines the status of one or more sockets, waiting if necessary, to perform synchronous I/O.

Script Commands:

DO_WSK_IsReadable()

DO_WSK_IsWriteable()

DO_WSK_Select()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAEFAULT(10014)	The Windows Sockets implementation was unable to allocate needed resources for its internal operations, or the readfds, writefds, exceptfds, or timeval parameters are not part of the user address space.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEINVAL(10022)	The time-out value is not valid, or all three descriptor parameters were null.
WSAEINTR(10004)	A blocking Windows Socket 1.1 call was canceled through

	WSACancelBlockingCall.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENOTSOCK(10038)	One of the descriptor sets contains an entry that is not a socket.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.
EINTR(4)	The select() function was interrupted before any of the selected events occurred and before the timeout interval expired. If SA_RESTART has been set for the interrupting signal, it is implementation-dependent whether select() restarts or returns with EINTR.
EINVAL(22)	An invalid timeout interval was specified.
EINVAL(22)	The nfds argument is less than 0, or greater than or equal to FD_SETSIZE.
EINVAL(22)	One of the specified file descriptors refers to a STREAM or multiplexer that is linked, directly or indirectly, downstream from a multiplexer.
EINVAL(22)	A component of the pointed-to time limit is outside the acceptable range: t_sec must be between 0 and 10**8, inclusive. t_usec must be greater than or equal to 0, and less than 10**6.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	An invalid file descriptor was given in one of the sets.
EINTR(4)	A non blocked signal was caught.
EINVAL(22)	n is negative, or the value contained within timeout is invalid.
ENOMEM(12)	Select was unable to allocate memory for internal tables.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_ERROR_00024

Fail to call ioctlsocket/ioctl function.

Description:

Fail to call ioctlsocket/ioctl function. The ioctlsocket/ioctl function controls the I/O mode of a socket.

Script Commands:

DO_WSK_Ioctlsocket()

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED(10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN(10050)	The network subsystem has failed.
WSAEINPROGRESS(10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENOTSOCK(10038)	The descriptor s is not a socket.
WSAEFAULT(10014)	The argp parameter is not a valid part of the user address space.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EFAULT(14)	The request argument requires a data transfer to or from a buffer pointed to by arg, but arg points to an illegal address.
EINVAL(22)	The request or arg argument is not valid for this device.
EIO(5)	Some physical I/O error has occurred.
ENOLINK(67)	The fildes argument is on a remote machine and the link to that machine is no longer active.
ENOTTY(25)	The fildes argument is not associated with a STREAMS device that accepts control functions.
ENXIO(6)	The request and arg arguments are valid for this device driver, but the

	service requested cannot be performed on this particular subdevice.
ENODEV(19)	The fildes argument refers to a valid STREAMS device, but the corresponding device driver does not support the ioctl() function.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	d is not a valid descriptor.
EFAULT(14)	argp references an inaccessible memory area.
ENOTTY(14)	d is not associated with a character special device.
ENOTTY(25)	The specified request does not apply to the kind of object that the descriptor d references.
EINVAL(22)	Request or argp is not valid.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WSK_WARNING_00025

Fail to call shutdown function.

Description:

Fail to call shutdown function. The shutdown function disables sends or receives on a socket.

Script Commands:

DO_WSK_Shutdown

Causes:

Windows

The following list describes the possible error codes returned by the WSAGetLastError function under Windows:

Error Code	Description
WSANOTINITIALISED (10093)	A successful WSAStartup call must occur before using this function.
WSAENETDOWN (10050)	The network subsystem has failed.

T WSAEINVAL (10022)	The how parameter is not valid, or is not consistent with the socket type. For example, SD_SEND is used with a UNI_RECV socket type.
WSAEINPROGRESS (10036)	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENOTCONN (10057)	The socket is not connected (connection-oriented sockets only).
WSAENOTSOCK (10038)	The descriptor is not a socket.

Solaris

The following list describes the possible error codes returned by the errorno function under Solaris:

Error Code	Description
EBADF(9)	s is not a valid file descriptor.
ENOMEM(12)	There was insufficient user memory available for the operation to complete.
ENOSR(63)	There were insufficient STREAMS resources available for the operation to complete.
ENOTCONN(134)	The specified socket is not connected.
ENOTSOCK(95)	s is not a socket.

Linux

The following list describes the possible error codes returned by the errorno function under Linux:

Error Code	Description
EBADF(9)	s is not a valid descriptor.
ENOTSOCK(88)	s is a file, not a socket.
ENOTCONN(107)	The specified socket is not connected.

Actions:

Check Winsock log file to get error number, then look at above table and find the description.

External Sources:

None

WWW

WWW Playback Error Codes

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

Error code	Description
WWW_ABORT_00001	Time out, value <timeout value> is invalid. Value must be between 30 and 65535, or 0 to reset to default.
WWW_ABORT_00002	<Virtual User> Crashed in the WWW Middleware.
WWW_ABORT_00003	Max browser thread, value <browser threads> is invalid. Max browser threads must be between 1 and 8, or 0 to reset to default.
WWW_ABORT_00004	Proxy version, value <Proxy version> is invalid. Proxy version must be 1.0 or 1.1.
WWW_ABORT_00005	HTTP version, value <HTTP version> is invalid. HTTP version must be 1.0 or 1.1.
WWW_ABORT_00006	Unable to open IPSpoof data pool.
WWW_ABORT_00007	Cannot execute SSL.
WWW_ABORT_00008	Parameter, value <parameter value> is invalid. <Brief description>
WWW_ERROR_00009	Page is not a redirect.
WWW_ABORT_00010	Failed to allocate memory for URL.
WWW_ABORT_00011	Memory allocation failure.
WWW_ABORT_00013	NON-SSL version of QALoad.
WWW_ABORT_00014	Attempted to assign value as SSL connect string.
WWW_ABORT_00015	Connect string must be in the form: CONNECT "server":"port"
WWW_ABORT_00016	Retry count, value <count number> is invalid. Retry count must be between 1 and 99 or 0 to reset to default.
WWW_ABORT_00017	Proxy authorization header would exceed maximum size.
WWW_ABORT_00018	Basic authorization header would exceed maximum size.
WWW_ABORT_00019	NTLM authorization unsupported on this platform.
WWW_ABORT_00020	Clear option, value <clear option> is invalid.
WWW_ABORT_00024	Unable to find User-Agent for SSL connect string.

WWW_ABORT_00025	Redirect received. Error in Original Request,
WWW_ERROR_00026	IP Address, value <IP address> is invalid.
WWW_ABORT_00036	Time out, value <time out value> is invalid. The time out value must be between 0 and 600 seconds
WWW_ERROR_00100	Generic exception: <Exception>
WWW_ABORT_00101	URL, value <url> is invalid.
WWW_ERROR_00102	Couldn't detect HTTP version.
WWW_ERROR_00103	Anchor tag, value <anchor name> not found.
WWW_ERROR_00104	Anchor tag, value <anchor index> not found.
WWW_ERROR_00106	Map region not found.
WWW_ERROR_00107	Form, value <form number> not found.
WWW_ABORT_00108	Form field, value <field type> is invalid.
WWW_ERROR_00109	Form field, value <field name> not found.
WWW_ERROR_00111	Cookie, value <cookie name> not found. <Brief description>
WWW_ERROR_00112	Page not found.
WWW_ERROR_00113	Response not found.
WWW_ERROR_00114	HTTP header, value <header name> not found.
WWW_ABORT_00116	Not enough space provided in output buffer. Needed: <space>
WWW_ERROR_00118	Response not found.
WWW_ERROR_00121	Failed to get script.
WWW_ERROR_00122	The client certificate file <client key file name>_cert.pem was not found in either the base QALoad or QALoad/Workbench directory.
WWW_ERROR_00123	The client key file <client key file name>_key.pem was not found in either the base QALoad or QALoad/Workbench directory.
WWW_ERROR_00124	Pattern string, value <pattern> not found. <Brief description>.
WWW_ERROR_00126	Unable to create the socket to <host name> (error =<error number>).
WWW_ERROR_00127	IP Spoofing unable to bind <spoofing IP address> (error = <error number>).
WWW_ERROR_00128	Unable to connect to <host name> (error = <error number>).

WWW_ERROR_00129	Handshake with SOCKS proxy server failed.
WWW_ERROR_00130	Send failed (error = <error number>).
WWW_ERROR_00131	Unable to connect to the server.
WWW_ERROR_00132	Timeout waiting for reply.
WWW_ERROR_00133	Exceeded maximum zero-length retries.
WWW_ERROR_00134	An error occurred while receiving connection <socket> (error = <error number>).
WWW_ABORT_00135	Unable to open binary file <file name>.
WWW_ERROR_00136	Unable to binary open file <file name> (error = <error number>).
WWW_ERROR_00137	Unable to stat file <file name> (error = <error number>).
WWW_ERROR_00138	Only read <byte read> bytes from <file> (expected <size>).
WWW_ERROR_00139	Failed to create SSL tunnel request.
WWW_ERROR_00140	Verification failed: <title> was received.
WWW_ERROR_00142	Verification failed. Second parameter is invalid.
WWW_ABORT_00143	HTTP Request is invalid. <Brief description>
WWW_ERROR_00160	Frame not found.
WWW_ERROR_00166	QALoad has exceeded the maximum number of retries.
WWW_WARNING_00300	Warning: Override IP Spoof filename <file> was not a valid data pool file, trying alternate source.
WWW_WARNING_00301	Warning: QALOAD_IPSPOOF environment variable <variable> was not a valid data pool file, trying alternate source.
WWW_WARNING_00302	Warning: Unsupported content encoding.
WWW_ABORT_00500	Generic message: Option <option> is invalid. <Brief description>
WWW_ABORT_00501	Generic message: Option <option>, value <value> is invalid. <Brief description>
WWW_ERROR_00502	Generic message: Option <option> not found. <Brief description>
WWW_ERROR_00503	Generic message: Option <option>, value <value> not found. <Brief description>
WWW_ABORT_01000	Get option <option> is invalid. <Brief description>
WWW_ABORT_01001	Get option <option>, value <value> is invalid. <Brief description>

WWW_ABORT_01100	Set option <option> is invalid. <Brief description>
WWW_ABORT_01101	Set option <option>, value <value> is invalid. <Brief description>
WWW_ABORT_01200	Specifier <option> is invalid. <Brief description>
WWW_ABORT_01201	Specifier <option>, value <value> is invalid. <Brief description>
WWW_ERROR_01202	Specifier <option> not found. <Brief description>
WWW_ERROR_01203	Specifier <option>, value <value> not found. <Brief description>
WWW_ABORT_01300	Click option <option> is invalid. <Brief description>
WWW_ABORT_01301	Click option <option>, value <value> is invalid. <Brief description>
WWW_ABORT_01400	Link option <option> is invalid. <Brief description>
WWW_ABORT_01401	Link option <option>, value <value> is invalid. <Brief description>
WWW_ERROR_01402	Link option <option> not found. <Brief description>
WWW_ERROR_01403	Link option <option>, value <value> not found. <Brief description>
WWW_ABORT_01500	Verify option <option> is invalid. <Brief description>
WWW_ABORT_01501	Verify option <option>, value <value> is invalid. <Brief description>
WWW_ERROR_01502	Verify option <option> not found. <Brief description>
WWW_ERROR_01503	Verify option <option>, value <value> not found. <Brief description>
WWW_ERROR_01600	Fill in control option <option> not found. <Brief description>
WWW_ERROR_01601	Fill in control option <option>, value <value> not found. <Brief description>
WWW_ERROR_03001	Content Check Violation: <Reason list>

WWW_ABORT_00001

The time out value set for the script is invalid.

Description:

The time out value set in the DO_SetTimeout call is not between the acceptable range of 30 to 65535 seconds. Setting the value to 0 sets the default time out value of 120 seconds.

Script Commands:

DO_SetTimeout

Causes:

An incorrect parameter was set in the DO_SetTimeout call.

Actions:

Modify the DO_SetTimeout call parameter (nTimeout) to a valid value.

External Sources:

None

WWW_ABORT_00002

The VU encountered a general protection fault (GPF) while executing the script.

Description:

The thread or process running the VU encountered a GPF while running the WWW script during playback.

Script Commands:

DO_SetRefreshTimeout

Causes:

- Code modifications introduced a bug into the compiled script.
- An environment incompatibility exists in the playback environment.

Actions:

- Ensure that any script modifications did not introduce a bug into the script.
- Ensure that the playback environment (OS, compiler, IE version) is correct for QALoad replay.
- Contact QALoad technical support if no scripting sources of the problem are found.

External Sources:

None

WWW_ABORT_00003

The max browser thread value set in the script is invalid.

Description:

The max browser thread value in the DO_SetMaxBrowserThreads call is not between 1 and 8, or 0 for the default value, which is 4.

Script Commands:

DO_SetMaxBrowserThreads

Causes:

An incorrect parameter was set in the DO_SetMaxBrowserThreads call.

Actions:

Modify the DO_SetMaxBrowserThreads call parameter (nCount) to a valid value.

External Sources:

None

WWW_ABORT_00004

The proxy version specified is invalid.

Description:

The proxy version specified in the Do_ProxyHttpVersion call or the Set [PROXY_HTTP_VERSION] call is not a supported version. Supported versions are "1.0" and "1.1".

Script Commands:

Do_ProxyHttpVersion

Set

Causes:

An incorrect parameter was set in the call.

Actions:

Modify the call parameter (szVersion) to a valid value.

External Sources:

None

WWW_ABORT_00005

The HTTP version specified is invalid.

Description:

The HTTP version specified in the DO_HTTPVersion call or the Set [HTTP_VERSION] call is not a supported version. Supported versions are "1.0" and "1.1".

Script Commands:

DO_HTTPVersion

Set

Causes:

An incorrect parameter was set in the call.

Actions:

Modify the call parameter (szVersion) to a valid value.

External Sources:

None

WWW_ABORT_00006

The IPSpoof data file cannot be opened for use with the script.

Description:

The data file specified in the DO_IPSpoofEnable call is not present or is not a valid datapool file. Datapool files use a comma-separated value file format.

Script Commands:

DO_IPSpoofEnable

Causes:

- The datafile specified does not exist or was not transferred to the player machine.
- The datafile specified in the script is incorrect.

Actions:

- Ensure that the IPSpoof datafile has been created and is specified correctly in the script.
- Create the IPSpooof file using Generate IPSpoof Datapool command in the Conductor. This file is called ipspoof.dat. Ensure the name is specified in the script.

External Sources:

None

WWW_ABORT_00007

The SSL library was not found on the playback machine.

Description:

The SSL code contained in this script cannot be executed. No SSL library was found on the playback machine.

Script Command:

DO_Https Click_On

Navigate_To XmlRequest

Post_To

Causes:

The playback machine does not have SSL library installed The SSL libraries must be installed for QALoad to playback an SSL script.

Actions:

Ensure that the SSL software is installed on the playback machine prior to running a script on the machine.

External Sources:

None

WWW_ABORT_00008

A required parameter for this call has been set to NULL.

Description:

A parameter for this call that requires an explicit value has been set to NULL. This command cannot execute unless this value is specified.

Script Commands:

DO_GetAnchorHREF	DO_SSLUseClientCert	DO_BlockTrafficFrom
DO_GetAnchorByNumber	DO_AddHeader	Navigate_To
DO_GetFormActionStatement	DO_SetAssumedContentType	XmlRequest
DO_GetFormValueByName	DO_ProxyAuthorization	Post_To

Set DO_BasicAuthorization DO_GetHeaderFromReply

DO_GetRedirectedURL DO_AllowTrafficFrom DO_GetCookie

Causes:

A NULL parameter for this script command was entered during script modification.

Actions:

Ensure that the parameter for this call in the script is a valid value.

External Sources:

None

WWW_ERROR_00009

The page requested to get the redirected URL does not actually redirect to a different URL.

Description:

The page that was requested did not redirect to a different URL. The request for the redirected URL is invalid for this request.

Script Commands:

DO_GetRedirectedURL

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- It is recommended that DO_DynamicRedirectHandling or select Dynamic Redirect Handling in the WWW options.
- Modify the script to remove this call from the script for this request.

External Sources:

None

WWW_ABORT_00010

Memory to store the redirected URL could not be allocated on the playback machine.

Description:

The playback machine could not allocate memory to store the redirected URL address for the request.

Script Commands:

DO_GetRedirectedURL

Causes:

- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.

External Sources:

None

WWW_ABORT_00011

Could not allocate memory on the playback machine when processing the API call.

Description:

Memory could not be allocated on the playback machine for the processing of the DO_UseProxy call.

Script Commands:

DO_Http	Navigate_To
DO_Https	Post_To
DO_UseProxy	Click_On
DO_AddHeader	XmlRequest

Causes:

- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.

External Sources:

None

WWW_ABORT_00013

The version of QALoad does not support SSL commands.

Description:

This installation of QALoad does not contain the SSL code required to playback these script commands.

Script Commands:

DO_SetSSLConnectString

DO_SSLUseCipher

Causes:

The appropriate SSL code was not installed on the playback machine.

Actions:

Ensure that the appropriate version of SSL is installed on the playback machine.

External Sources:

None

WWW_ABORT_00014

The value sent as the connect message to the API call is not valid.

Description:

The szConnectMessage parameter for the DO_SetSSLConnectString does not exist or is not a valid connect message to enable SSL on the server.

Script Commands:

DO_SetSSLConnectString

Causes:

Script modification resulted in an incorrect parameter passed to the API call.

Actions:

Ensure that the DO_SSLSetConnectString contains a valid parameter to enable SSL on the server.

External Sources:

None

WWW_ABORT_00015

The SSL connect string is not in the correct format for enabling SSL on the server.

Description:

The connect string passes as the szConnectString parameter to the DO_SSLSetConnectString call is not in the format: CONNECT <server> <port>

Script Commands:

DO_SetSSLConnectString

Causes:

Script modification resulted in an incorrect parameter passed to the API call.

Actions:

Ensure that the DO_SSLSetConnectString contains a valid parameter to enable SSL on the server.

External Sources:

None

WWW_ABORT_00016

The maximum retry count specified for the script is invalid.

Description:

The maximum WWW request retry count specified as a parameter (value) for the DO_SetMaximumRetries call is not between 1 to 99, or is not set to 0 to set the default, which is 4.

Script Commands:

DO_SetMaximumRetries

Causes:

Script modification resulted in an invalid parameter passed to theDO_SetMaximumRetries call.

Actions:

Ensure that the DO_SetMaximumRetries call contains a valid parameter to set the maximum retries for WWW requests.

External Sources:

None

WWW_ABORT_00017

The header specified for the proxy authorization is too large.

Description:

The proxy authorization header created from encoding the username and password is longer than the expected header sent as part of the DO_ProxyAuthorization call.

Script Commands:

DO_ProxyAuthorization

Causes:

Script modification resulted in an invalid username or password passed to the DO_ProxyAuthorization call.

Actions:

Ensure that the DO_ProxyAuthorization call contains the correct username and password for the proxy.

External Sources:

None

WWW_ABORT_00018

The header specified for basic authorization is too large.

Description:

The authorization header created from encoding the username and password is longer than the expected header sent as part of the DO_BasicAuthorization call.

Script Commands:

DO_BasicAuthorization

Causes:

Script modification resulted in an invalid username or password passed to the DO_BasicAuthorization call.

Actions:

Ensure that the DO_BasicAuthorization call contains the correct username and password for authorization.

External Sources:

None

WWW_ABORT_00019

This platform running the playback script does not support NTLM authorization.

Description:

The playback machine is running on a non-Windows platform and does not support NTLM authorization with the server.

Script Commands:

DO_NTLMAuthorization

Causes:

- The script is being run on a different platform than the platform where the script was recorded.
- The script was modified to include NTLM authorization and run on non-Windows platform.

Actions:

- Ensure that the correct playback machine was selected for the script.
- Disable the NTLM authorization API call if not applicable for script playback.

External Sources:

None

WWW_ABORT_00020

An invalid option was specified to be cleared.

Description:

The parameter specifying the option to be cleared for the Clear or DO_Clear command is not a valid WWW option. The Function Wizard enumerates the valid parameter values for this call.

Script Commands:

DO_Clear

Clear

Causes:

Script modification resulted in an incorrect parameter passed to the DO_Clear/Clear call.

Actions:

Ensure the parameter (nType) passed to the DO_Clear/Clear call is correct.

External Sources:

None

WWW_ABORT_00024

Could not find User-Agent information set to use for SSL connect string.

Description:

A dynamic redirect to a SSL site failed because there was no User-Agent specified in a previous proxy authorization call.

Script Commands:

DO_Https Click_On

Navigate_To XmlRequest

Post_To

Causes:

- Script modification resulted in an SSL redirect request without a prior proxy authorization API call.
- The WWW site behavior changed from the time the session was recorded.

Actions:

- Ensure that the appropriate call for Proxy Authorization is added into the script.
- Ensure that the behavior of the WWW sites is as expected by doing another session record.

External Sources:

None

WWW_ABORT_00025

An unexpected redirect was received from the WWW server because it could not process the URL sent.

Description:

A redirect was received from the WWW server because it could not process the URL specified in the WWW request.

Script Commands:

DO_Http Post_To

Click_On XmlRequest

Navigate_To

Causes:

- Script modification resulted in a WWW request with an invalid URL address specified.
- The WWW site behavior changed from the time the session was recorded.

Actions:

- Ensure that the appropriate URL address was specified for the DO_Http call.
- Ensure that the behavior of the WWW sites is as expected by doing another session record.

External Sources:

None

WWW_ERROR_00026

IP Address, value <IP address> is invalid.

Description:

The IP address specified is not a valid IP spoof address. IP spoofing cannot be enabled, as this address is not valid for the playback machine.

Script Commands:

DO_IPSpoofEnable

Causes:

- One or more of the IP addresses specified in the IP spoof datapool file are incorrect and do not correspond with the IP addresses of the machine specified for playback.

Actions:

- Check to see that the IP addresses specified in the IP spoof datapool file are correct and correspond to the IP addresses for the playback machine.

External Sources:

None

WWW_ABORT_00036

The refresh timeout value set for the script is invalid.

Description:

The timeout value set in the DO_SetRefreshTimeout call is not between the acceptable range of 30 to 65535 seconds. Setting the value to 0 sets the default timeout value of 120 seconds.

Script Commands:

DO_SetRefreshTimeout

Causes:

An incorrect parameter was set in the DO_SetRefreshTimeout call.

Actions:

Modify the DO_SetRefreshTimeout call parameter (nTimeout) to a valid value.

External Sources:

None

WWW_ERROR_00100

An exception occurred during processing the WWW API call.

Description:

An exception was thrown while processing the API call. The error string indicates the nature of the exception thrown.

Script Commands:

DO_Http	Navigate_To
DO_Https	Post_To
DO_GetAnchorHREF	XmlRequest
DO_GetAnchorByNumber	Set
DO_GetRedirectedURL	Click_On
DO_ProxyExceptions	

Causes:

- Script modification resulted in an abnormal condition encountered by the API call.

- An unexpected condition resulted while executing the script.

Actions:

- The playback machine may not be a suitable environment for the number of VUs specified for the load test.
- Script modifications may have introduced a bug to the script.
- Contact QALoad technical support if not cause can be found.

External Sources:

None

WWW_ABORT_00101

URL, value <url> is invalid.

Description:

The parameter passed to the API call is not a valid URL address. The URL address could not be processed.

Script Commands:

DO_Http Navigate_To

DO_Https Post_To

Click_On XmlRequest

Causes:

- Script modification resulted in an invalid value passed to the API call.

Actions:

- Modify the API call parameter (URL) to a valid URL address.

External Sources:

None

WWW_ERROR_00102

The HTTP version could not be detected from the WWW page request.

Description:

A valid HTTP version was not specified in the header of the HTTP request. Valid versions of the HTTP header are "1.0" and "1.1".

Script Commands:

DO_Http Navigate_To

DO_Https Post_To

Click_On XmlRequest

Causes:

The HTTP Version was not set for the WWW page request..

Actions:

Modify the script to ensure the HTTP version is set for this request. Use the Set command (Visual Script) or the DO_HTTPProxy command (EZScript) to set the HTTP version for the page request. Valid versions of the HTTP header are "1.0" and "1.1".

External Sources:

None

WWW_ERROR_00103

The specified anchor tag was not found in the HTML document returned by the last page request.

Description:

The anchor tag specified by the name parameter to the DO_GetAnchorHREF call was not among the anchor tags found in the HTML document returned form the last request.

Script Commands:

DO_GetAnchorHREF

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid value passed as the anchor tag parameter to the DO_GetAnchorHREF call.

Actions:

- Modify the DO_GetAnchorHREF statement to find the correct name of the anchor tag in the HTML document reply.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00104

The specified anchor identified as the nth anchor tag was not found in the HTML document returned by the last page request.

Description:

The anchor specified as the nth anchor tag parameter to the DO_GetMapHREF call was not found in the HTML document returned from the last request. The page request sent a smaller number of anchor tags than the number specified in the DO_GetAnchorByNumber call.

Script Commands:

DO_GetAnchorByNumber

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid value passed as the anchor number parameter to the DO_GetAnchorByNumber call.

Actions:

- Modify the DO_GetAnchorByNumber statement to find the correct anchor tag by number in the HTML document reply.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00106

The region identified as the nth region was not found in the HTML document returned by the last page request.

Description:

The region specified as the nth region parameter to the DO_GetClientMapHREF call was not found in the HTML document returned from the last request. The page request sent a smaller number of regions than the number specified in the DO_GetClientMapHREF call.

Script Commands:

DO_GetClientMapHREF

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid value passed as the map tag parameter to the DO_GetClientMapHREF call.

Actions:

- Modify the DO_GetClientMapHREF statement to find the region in the HTML document reply.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00107

The specified form identified as the nth form tag was not found in the HTML document returned by the last page request.

Description:

The form specified as the nth form tag parameter to the API call was not found in the HTML document returned from the last request. The page request sent a smaller number of form tags than the number specified in the API call.

Script Commands:

DO_GetFormActionStatement

DO_GetFormValueByName

Causes:

- The WWW site has changed its behavior from the time the session was recorded.

- Script modification resulted in an invalid value passed as the form number parameter to the API call.

Actions:

- Modify the API call to find the correct form tag by number in the HTML document reply.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ABORT_00108

Form field, value <field type> is invalid.

Description:

The form type specified for the DO_GetFormValueByName command must be one of the valid form types for the HTML DOM model supported by QALoad.

Script Commands:

DO_GetFormValueByName

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The recorded session captured a form value that is not supported by QALoad playback.

Actions:

- Modify the form type value parameter to a valid form type supported by the QALoad HTML DOM model.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_00109

The field name specified was not found in the form specified.

Description:

The field specified as the third (FieldName) parameter was not found in the form specified as the second (FormName) parameter for the DO_GetFormValueByName call.

Script Commands:

DO_GetFormValueByName

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid value passed as the field name parameter to the DO_GetFormValueByName call.

Actions:

Modify the field name parameter to a valid field in the form specified for the DO_GetFormValueByName call.

External Sources:

None

WWW_ERROR_00111

The response header returned from the last request did not contain the a cookie specified.

Description:

The last reply did not contain the cookie specified by name as the first parameter for the DO_GetCookieFromReplyEx call.

Script Commands:

DO_GetCookieFromReplyEx

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid first parameter for the DO_GetCookieFromReplyEx call for this page request.

Actions:

- Remove the API call from the script.
- Modify the DO_GetCookieFromReplyEx call specifying the correct cookie name as the first parameter.

External Sources:

None

WWW_ERROR_000112

The WWW server did not return a valid HTTP page to extract the HTTP error number.

Description:

A valid HTTP page was not returned in the response from the last HTTP request. The server HTTP response cannot be extracted from the reply.

Script Commands:

DO_GetLastHttpError

Causes:

The WWW server could not process the last HTTP request properly.

Actions:

- Ensure the WWW server is available and able to process requests.
- Ensure the last page request is a valid request.

External Sources:

None

WWW_ERROR_00113

Response not found.

Description:

The response from the server was not found because the server has not sent any data that has been stored.

Script Commands:

DO_VerifyDocTitle

Causes:

- The WWW server did not send a response that could be stored by QALoad.

Actions:

- Ensure the WWW server is available and able to process requests and send a response back.
- Ensure the last page request is valid and that the WWW server understands it.

External Sources:

None

WWW_ERROR_00114

No HTTP response was received form the server to extract the header value.

Description:

The server did not send a response for the last page request. There is no HTTP header to retrieve the header value.

Script Commands:

DO_GetHeaderFromReply

Causes:

The WWW server could not process the last HTTP request properly.

Actions:

- Ensure that the WWW server is available and able to process requests.
- Ensure that the last page request is a valid request.

External Sources:

None

WWW_ABORT_00116

Not enough space provided in output buffer. Needed: <space>.

Description:

The size of the destination buffer is not big enough to hold the result from the API command.

Script Commands:

DO_GetHeaderFromReply

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Record another session to get the WWW site's current behavior.
- Modify the API command to accommodate for the small buffer by providing a bigger one.

External Sources:

None

WWW_ERROR_00118

The last request was not received. The reply buffer could not be extracted.

Description:

The response for the last request was not received. This indicates there was a problem with the last request.

Script Commands:

DO_GetReplyBuffer

Causes:

The WWW server could not process the last HTTP request properly.

Actions:

- Ensure the WWW server is available and able to process requests.
- Ensure the last page request is a valid request.

External Sources:

None

WWW_ERROR_00121

A valid proxy configuration script was not found at the URL address.

Description:

The URL address specified as the first parameter in the DO_UseProxyAutomaticConfiguration call did not return a valid proxy configuration script.

Script Commands:

DO_UseProxyAutomaticConfiguration

Causes:

- The proxy configuration server is not available or could not process the last request properly.
- Script modification resulted in an invalid first parameter for the DO_UseProxyAutomaticConfiguration call for this page request.

Actions:

Ensure that the proxy configuration server is available and able to process proxy configuration requests.

External Sources:

None

WWW_ERROR_00122

The client certificate filename created from the name parameter could not be found in the QALoad or QALoad>Workbench directories.

Description:

The client certificate filename, constructed from the first (szName) parameter of the DO_SSLUseClientCert call and the "_cert.pem" string, could not be found in either the base QALoad or the QALoad>Workbench directories. This certificate must be present in either of these file locations for the call to extract the file for use in playback.

Script Commands:

DO_SSLUseClientCert

Causes:

- The client certificate was not created or located in the expected directories, or it was moved from the expected directory prior to playback.
- Script modification resulted in an invalid first parameter for the name parameter for the DO_SSLUseClientCert call.

Actions:

- Ensure that the client certificate file has been created and is available in one of the two expected directories, either the base QALoad directory or the QALoad>Workbench directory.
- Ensure that the name parameter passed to the DO_SSLClientCert call is valid, so that the string "_cert.pem" concatenated to it identifies the correct client certificate.

External Sources:

None

WWW_ERROR_00123

The client certificate key file name created from the name parameter could not be found in the QALoad or QALoad>Workbench directories.

Description:

The client certificate key filename, constructed from the first (szName) parameter of the DO_SSLUseClientCert call and the "_key.pem" string, could not be found in either the base QALoad or the QALoad>Workbench directories. This certificate key must be present in either of these file locations for the call to extract the key for use in playback.

Script Commands:

DO_SSLUseClientCert

Causes:

- The client certificate key file was not created or located in the expected directories, or it was moved from the expected directory prior to playback.

- Script modification resulted in an invalid first parameter for the name parameter for the DO_SSLUseClientCert call.

Actions:

- Ensure that the client certificate file has been created and is available in one of the two expected directories, either the base QALoad directory or the QALoad>Workbench directory.
- Ensure that the name parameter passed to the DO_SSLClientCert call is valid, so that the string "_cert.pem" concatenated to it identifies the correct client certificate.

External Sources:

None

WWW_ERROR_00124

The identifying left (pre-) string delimiter to extract a string value was not found in the HTML document reply.

Description:

The left-side string delimiter passed in as the second parameter to the DO_GetUniqueStringEx call was not found in the HTML document reply.

Script Commands:

DO_GetUniqueStringEx

Causes:

- The WWW server sent an unexpected reply.
- The left-side delimiter string specified for the DO_GetUniqueStringEx call is not correct.
- The WWW site has changed its behavior from the time the session was recorded.

Actions:

- Modify the DO_GetUniqueStringEx call to the correct left-side delimiter text identified in the HTML response.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00126

The replay machine could not create a socket for communication with the server.

Description:

The socket to initiate communication with the WWW server could not be created on the playback machines.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The playback machine may have exhausted all available sockets.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

Ensure the playback machine has enough free resources to run the number of VUs allocated to the player on this machine. If this error is occurring on a Windows player, registry settings can be adjusted to both increase the maximum number of sockets and to reduce the time wait delay sockets go through after they have been shutdown.

To increase the maximum number of sockets:

- Run regedit.exe.
- Create the MaxUserPort key in the registry at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters.
- Set the value of MaxUserPort to be large value, 30,000 or more. NOTE: the maximum value is 65534.

To reduce the time wait delay:

- Run regedit.exe.
- Create the TcpTimedWaitDelay key in the registry at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters.
- Set the value of TcpTimedWaitDelay to 30.

External Sources:

- Microsoft Knowledge Base Article 149532
- Microsoft Knowledge Base Article 196271
- Microsoft Knowledge Base Article 328476
- Microsoft Knowledge Base Article 319502

WWW_ERROR_00127

The playback machine could not bind a socket with the IP address specified.

Description:

The playback machine could not bind a socket to the static IP address specified in a previous DO_IPSpoofEnable or Set command.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

The value specified as a spoof IP address is not associated with a network card on the playback machine.

Actions:

Ensure that the playback machine has been set up with the static IP address associated with the value specified in the DO_IPSpoofEnable or Set command.

External Sources:

None

WWW_ERROR_00128

The socket was unable to connect to the server to initiate a WWW session.

Description:

The socket to initiate communication with the WWW server could not set up a connection.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server may not be available and receiving socket connections.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for it's specifications.

Actions:

- Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure that the WWW server is available and able to process requests.

External Sources:

None

WWW_ERROR_00129

The playback machine was unable to complete a handshake with the SOCKS proxy server.

Description:

The attempt to complete a handshake (connect) with the SOCKS proxy machine was not successful.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The proxy configuration server is not available or could not process the last request properly.
- Script modification resulted in an invalid URL or IP address specified as the proxy server in a previous API call.

Actions:

- Ensure the proxy configuration server is available and able to process proxy configuration requests.
- Ensure that a valid URL or IP address is specified as the proxy server in any previous API calls.

External Sources:

None

WWW_ERROR_00130

The attempt to send the page request failed.

Description:

The last send request failed. The error number is specified in the message.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server may not be available and receiving socket connections.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

- Ensure the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure the WWW server is available and able to process requests.

External Sources:

None

WWW_ERROR_00131

The socket was unable to connect to the server to send the last page request.

Description:

The socket to initiate communication with the WWW server could not set up a connection to send the last page request.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server is extremely low on resources.

- The network may be overloaded with traffic.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

- Ensure the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure the WWW server is available and able to process requests.

External Sources:

None

WWW_ERROR_00132

The reply for the last request was not received within the timeout value specified.

Description:

The WWW server did not return a reply message to the last request within the timeout value set for the page request.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server is extremely low on resources.
- The network may be overloaded with traffic.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.
- The timeout value set in the script is too low.

Actions:

- Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure that the WWW server is available and able to process requests.
- Increase the timeout value parameter of the appropriate Set (Visual Script) or DO_SetTimeout (EZScript) commands. Use the Function Wizard for help in scripting.

External Sources:

None

WWW_ERROR_00133

The number of zero-length replies from the server for the last request exceeded the value set as the maximum retry limit.

Description:

The WWW server returned more zero-length retries for the last request than the maximum zero-length retries set for the script.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server is extremely low on resources.
- The network may be overloaded with traffic.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

- Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure that the WWW server is available and able to process requests.
- Increase the maximum zero-length retry value parameter of the appropriate Set (Visual Script) or DO_SetMaximumRetries (EZScript) commands. Use the Function Wizard for help in scripting.

External Sources:

None

WWW_ERROR_00134

A socket error occurred when receiving the data on the specified connection.

Description:

The connection received a socket error when receiving the reply for the last request sent out on that socket connection.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server is extremely low on resources.
- The network may be overloaded with traffic.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

- Ensure the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure the WWW server is available and able to process requests.
- Increase the maximum zero-length retry value parameter of the appropriate Set (Visual Script) or DO_SetMaximumRetries (EZScript) commands. Use the Function Wizard for help in scripting.

External Sources:

None

WWW_ABORT_00135

Unable to open binary file .

Description:

The filename specified in the error message could not be opened as a binary file during the response form the last page request.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The file sent may be corrupted.
- There is a problem on the WWW server machine.

Actions:

- Ensure the integrity of the WWW server environment.
- Ensure the specified file is not corrupt on the server.
- Ensure the integrity of the network.

External Sources:

None

WWW_ERROR_00136

The specified file sent as part of the response for the last request could not be opened.

Description:

The filename specified in the error message could not be opened as a binary file during the response form the last page request. The error number is in the error message.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The file sent may be corrupted.
- There is a problem on the WWW server machine.

Actions:

- Ensure the integrity of the WWW server environment.
- Ensure that the specified file is not corrupt on the server.
- Verify the integrity of the network.

External Sources:

None

WWW_ERROR_00137

The specified file sent as part of the response for the last request could not be queried with the stat command.

Description:

The filename specified in the error message could not be queried by the stat command during the response from the last page request. The error number returned by the stat command is in the error message.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The file sent may be corrupted.
- There is a problem on the WWW server machine.

Actions:

- Ensure the integrity of the WWW server environment.
- Ensure that the specified file is not corrupt on the server.
- Ensure the integrity of the network.

External Sources:

None

WWW_ERROR_00138

The specified file sent as part of the response for the last request differed in size from the expected size value

Description:

The actual size of the filename specified in the error message is not the expected size of the filename from the WWW server response.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The file sent may be corrupted.
- There is a problem on the WWW server machine.

Actions:

- Ensure the integrity of the WWW server environment.
- Ensure that the specified file is not corrupt on the server.
- Ensure the integrity of the network.

External Sources:

None

WWW_ERROR_00139

The SSL tunnel request could not be created.

Description:

The request to create an SSL tunnel with the SSL server failed. The request could not be created.

Script Commands:

DO_Https

Causes:

- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.

External Sources:

None

WWW_ERROR_00140

The document title received in the last request does not match the expected title.

Description:

The document title parsed from the server response does not match the first parameter (szTitle) passed to the DO_VerifyDocTitle call.

Script Commands:

DO_VerifyDocTitle

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- The WWW server does not send a consistent document title as part of the response to this request.
- Script modification resulted in an invalid value passed as the first parameter (szTitle) to the DO_VerifyDocTitle call.

Actions:

- Modify the first parameter (szTitle) of the DO_VerifyDocTitle call to the expected document title name in the HTML document reply.
- Remove the DO_VerifyDocTitle call if the server response is not expected to be consistent.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00142

The document title comparison type specified is not valid.

Description:

The document title comparison type specified as the second parameter to the DO_VerifyDocTitle call is not valid. Valid values for this parameter are TITLE (match full title), PREFIX (match the first part of the title) or SUFFIX (match the end part of the title).

Script Commands:

DO_VerifyDocTitle

Causes:

Script modification resulted in an invalid value passed as the second parameter (nType) to the DO_VerifyDocTitle call.

Actions:

- Modify the second parameter (ntype) of the DO_VerifyDocTitle call to a valid document title type comparison enumerated value (TITLE, PREFIX, or SUFFIX).
- Remove the DO_VerifyDocTitle from the script.

External Sources:

None

WWW_ABORT_00143

HTTP request is invalid. <Brief description>

Description:

The Request string specified for the API command is invalid.

Script Commands:

DO_Http DO_Https

Navigate_To XmlRequest

Post_To Click_On

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- Use of variables that are set at runtime, which could contain unexpected data.

Actions:

- Ensure that a valid parameter string value is passed to the API command. E.g. Check for the correctness of each component in the parameter string value such as the request method, URL, HTTP version, HTTP header fields.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.

External Sources:

None

WWW_ERROR_00160

The frame referenced by the Get call could not be found in the current HTML document.

Description:

The frame referenced by the Get call (with the FRAME option as the first parameter) could not be found based on the parameters (specifier, description, and/or count) set for the Get call.

Script Commands:

Get

Causes:

- The WWW site has changed its behavior from the time the session was recorded.
- Script modification resulted in an invalid parameter set for the Get call.

Actions:

- Ensure that the parameters passed to the Get call are correct.
- Record another session to get the WWW site's current behavior.

External Sources:

None

WWW_ERROR_00166

The number of page request retries from the server for the last request exceeded the value set as the connection retries limit.

Description:

The number of page request retries for the last HTTP request was greater than the connection retry limit set for the script.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The WWW server is extremely low on resources.
- The network may be overloaded with traffic.
- The system is extremely low on resources.
- The playback machine is running more VUs than is proper for its specifications.

Actions:

- Ensure that the playback machine has enough free resources to run the number of VUs allocated to player on this machine.
- Ensure that the WWW server is available and able to process requests.
- Increase the maximum zero-length retry value parameter of the appropriate Set (Visual Script) or DO_SetMaximumRetries (EZScript) commands. Use the Function Wizard for help in scripting.

External Sources:

None

WWW_WARNING_00300

The IP spoof file is not a valid data pool file.

Description:

The IP spoof file specified cannot be opened as a datapool file for the extraction of IP addresses.

Script Commands:

DO_IPSpoofEnable

Causes:

- The IP spoof file does not exist in the datapool folder.
- The IP spoof file is not formatted as a datapool file.
- The IP spoof file is not formatted for IP spoofing.

Actions:

- QALoad will try an alternate source to get IP spoof data.
- Ensure that the IP spoof file exists in the datapool directory (or other specified directory), that it is a valid QALoad datapool file, and that it has correct IP spoof values and format.

External Sources:

None

WWW_WARNING_00301

The IP spoof environment variable does not refer to a valid datapool file.

Description:

The IP spoof file specified by an environment variable cannot be opened as a datapool file for the extraction of IP addresses.

Script Commands:

DO_IPSpoofEnable

Causes:

- The environment variable was set incorrectly and does not point to the correct IP spoof file.
- The IP spoof file does not exist in the datapool folder.
- The IP spoof file is not formatted as a datapool file.
- The IP spoof file is not formatted for IP spoofing.

Actions:

- QALoad will try an alternate source to get IP spoof data.
- Ensure that the environment variable is set to point to the correct IP spoof file.
- Ensure that the IP spoof file exists in the datapool directory (or other specified directory), that it is a valid QALoad datapool file, and that it has correct IP spoof values and format.

External Sources:

None

WWW_WARNING_00302

The content encoding specified in the last response is not supported by QALoad.

Description:

The last WWW page had a content encoding in the header that is not supported by QALoad. This may affect the playback behavior.

Script Commands:

Post_To	Click_On
Navigate_To	DO_Http
XmlRequest	DO_Https

Causes:

- The WWW server has changed since the script was recorded.
- The WWW server supports content that QALoad does not support.

Actions:

No action need be taken, but this may indicate a change in WWW server behavior that may affect playback.

External Sources:

None

WWW_ABORT_00500

Generic message: Option <option> is invalid. <Brief description>.

Description:

The API option passed to the API command is invalid.

Script Commands:

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.

External Sources:

None

WWW_ABORT_00501

Generic message: Option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API option is invalid.

For example, if an API command is expecting a string value and a NULL is passed to the API command, this exception could be thrown.

Script Commands:

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- Use of variables that are set at runtime, which could contain unexpected data.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.

External Sources:

None

WWW_ERROR_00502

Generic message: Option <option> not found. <Brief description>.

Description:

The API option passed to the API command was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Causes:

- Script modification resulted in an invalid option passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ERROR_00503

Generic message: Option <option>, value <value> not found. <Brief description>.

Description:

The parameter value described by the API option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ABORT_01000

Get option <option> is invalid. <Brief description>.

Description:

The API option passed to the Get API command is invalid.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the Get API command.

External Sources:

None

WWW_ABORT_01001

Get option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API option is invalid for the Get API command.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the Get API command.

External Sources:

None

WWW_ABORT_01100

Set option <option> is invalid. <Brief description>.

Description:

The API option passed to the Set API command is invalid.

Script Commands:

Set

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid option value is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the Set API command.

External Sources:

None

WWW_ABORT_01101

Set option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API option is invalid for the Set API command.

Script Commands:

Set

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid option type is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the Set API command.

External Sources:

None

WWW_ABORT_01200

Specifier <option> is invalid. <Brief description>.

Description:

The API Specifier option is invalid for the API command.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ABORT_01201

Specifier <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API Specifier option is invalid for the API command.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_01202

Specifier <option> not found. <Brief description>.

Description:

The API Specifier option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid option passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid API option is used.
- Ensure that the server is responding with the expected data.

- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ERROR_01203

Specifier <option>, value <value> not found. <Brief description>.

Description:

The parameter value described by the API Specifier option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Get

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ABORT_01300

Click option <option> is invalid. <Brief description>.

Description:

The API Click_On option is invalid for the Click_On API command.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.

- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ABORT_01301

Click option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API Click_On option is invalid for the Click_On API command.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ABORT_01400

Link option <option> is invalid. <Brief description>.

Description:

The API Link option is invalid for the API command.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ABORT_01401

Link option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API Link option is invalid for the API command.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_01402

Link option <option> not found. <Brief description>.

Description:

The API Link option was not found in the parsed and or unparsed reponse(s) from the server.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid option passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ERROR_01403

Link option <option>, value <value> not found. <Brief description>.

Description:

The parameter value described by the API Link option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Click_On

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ABORT_01500

Verify option <option> is invalid. <Brief description>.

Description:

The API Verify option is invalid for the API Verify command.

Script Commands:

Verify

Causes:

- Script modification resulted in an invalid option passed to the API command.

Actions:

- Ensure that a valid API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ABORT_01501

Verify option <option>, value <value> is invalid. <Brief description>.

Description:

The parameter value described by the API Verify option is invalid for the API Verify command.

Script Commands:

Verify

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_01502

Verify option <option> not found. <Brief description>.

Description:

The API Verify option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Verify

Causes:

- Script modification resulted in an invalid option passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ERROR_01503

Verify option <option>, value <value> not found. <Brief description>.

Description:

The parameter value described by the API Verify option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Verify

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_01600

Fill in control option <option> not found. <Brief description>.

Description:

The API Fill_In option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Fill_In

Causes:

- Script modification resulted in an invalid option passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid API options for the API command.

External Sources:

None

WWW_ERROR_01601

Fill in control option <option>, value <value> not found. <Brief description>.

Description:

The parameter value described by the Fill_In option was not found in the parsed and or unparsed response(s) from the server.

Script Commands:

Fill_In

Causes:

- Script modification resulted in an invalid parameter value passed to the API command.
- The server is not sending the same expected data that was received during capture. Most likely the WWW site has changed.

Actions:

- Ensure that a valid parameter value described by the API option is used.
- Ensure that the server is responding with the expected data.
- Use the description that is given in the error message at runtime to help you troubleshoot the problem.
- Use the API command reference to find details about valid parameter values for the API command.

External Sources:

None

WWW_ERROR_03001

One or more Content Checks failed.

Description:

During WWW playback, a required content string was not found or a prohibited content string was detected. It is possible that multiple content check violations have occurred. If multiple content check violations have occurred, then the different violation will be separated by semicolons.

Script Commands:

Post_To Click_On

Navigate_To DO_Http

XmlRequest DO_Https

Causes:

- The WWW server had an error and returned an error page.
- The script is not correctly parameterized.
- The network may be overloaded with traffic and an incomplete page was returned.

Actions:

- Check the server to see if any error have occurred on it. Checking the condition of a web server varies from server to server, please refer to your server's documentation.
- Either the content check or the data sent to the server with in Fill_In, Set CGI_PARAMETER, Set POST_DATA, or Set POST_FILE needs to be properly parameterized correctly.
- Check with the network administrator to see if the network has been overloaded during the performance test.

External Sources:

None

Customer Support

Compuware customer support

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

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

Contact information

Compuware Corporation
One Campus Martius
Detroit, MI 48226-5099

(800) 538-7822

World Wide Web Information

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

FrontLine Support Web Site

You can access online technical support for Compuware products using our FrontLine support Web site. FrontLine provides fast access to critical information about your QACenter product. You can read or download documentation, frequently asked questions, and product fixes, or email your questions or comments. To access FrontLine, you must first register and obtain a password. To register, point your browser at http://frontline.compuware.com.

ZipFile Wizard

You can use the ZipFile Wizard to transmit smaller files to QALoad Technical Support. This simplifies the task of collecting the files needed to assist you in resolving your problem.

Overview of the ZipFile Wizard

The ZipFile Wizard simplifies the task of collecting the files needed by the Technical Support team to analyze and resolve a problem that you might be having with QALoad. It facilitates navigation through QALoad's installation so you can easily select the required files without actually knowing where they reside. Once selected, the files are collected and compressed into a ZIP file that optionally can be transmitted to the QALoad Technical Support team.

Next >>

Accessibility

Accessibility features

Compuware is committed to compliance with Section 508 standards for accessibility in software products. Section 508 standards were enacted by Congress in 1998 as an amendment to the Rehabilitation Act. The standards require federal agencies to increase the accessibility of their electronic information to people with disabilities. Software that is fully compliant with Section 508 standards provides information access to people with disabilities that is comparable to the information access available to people without disabilities. Software that is compliant with exceptions, such as QALoad, provides support for most, but not all, criteria of compliance.

This topic describes the level of accessibility currently available in QALoad, including the product documentation.

Accessibility of QALoad components

All components of QALoad — Script Development Workbench, Conductor, Player, and Analyze — support the basic criteria of accessibility for software applications. For most categories of compliance, there are exceptions in which assistive technologies do not work in all situations or the alternate methods of information retrieval or product function are not always available. However, the following basic accessibility features are available in QALoad:

- Product functions are executable from the keyboard
- The application does not disrupt or disable activated features of other products that are identified as accessibility features
- The product provides an on-screen indication of the current focus and the focus can be tracked by assistive technologies
- The identity, operation, and state of elements of the user interface are available to assistive technologies
- Textual information is provided through the operating system functions for displaying text
- The application does not override user-selected contrast and color selections
- Color-coding is not used as the only means of conveying information
- Electronic forms allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form

Accessibility of QALoad documentation

Generally, the QALoad documentation is accessible and can be read by screen readers. Alternate formats of the documentation are not currently available.

Online help

QALoad's online help system is HTML-based and can be read by screen readers. All graphics in the online help have text descriptions. Topics in the online help can be printed.

Release Notes

The QALoad Release Notes is an HTML document that can be read by screen readers and printed.

Installation Guide

The QACenter Performance Edition Installation and Configuration Guide is an accessible PDF-based book that provides text descriptions for images in addition to the ability to be read by some screen readers and the

Read Aloud feature of Adobe Acrobat. To take advantage of the accessibility features in the Installation Guide, you must have Adobe Acrobat Reader 6.0 or later. This book can be printed.

Assistive technology tools that enhance the accessibility of QALoad

There are many third-party assistive technology tools that you can use to access the QALoad user interface and product documentation by alternate means. These products perform a variety of functions, such as enlarging the user interface and reading text on the interface or in the documentation. Many features are available in Windows, such as:

- Windows Narrator
- Windows accessibility wizard (includes features such as SoundSentry, ShowSounds, StickyKeys, and MouseKeys)
- Windows On-Screen Keyboard
- Microsoft Magnifier

Refer to the Windows documentation to learn more about using these features. Most other tools fall into one of the following two categories:

- **Screen readers**: Software programs that present graphics and text as speech. A screen reader is used to verbalize, or "speak," everything on the screen including names and descriptions of control buttons, menus, text, and punctuation. (Example: JAWS for Windows by Freedom Scientific, Inc.)
- **Screen enlargers (or screen magnifiers)**: Software that works like a magnifying glass on other applications. They enlarge a portion of the screen as the user moves the focus — increasing legibility for some users. Some screen enlargers allow you to zoom in and out on a particular area of the screen. (Example: ZoomText Magnifier by AI Squared)

Product shortcut keys

The following tables list the defined shortcut keys for each component of QALoad. You can use these key combinations to open dialog boxes, start or stop processes, or interact in other ways with QALoad without using a mouse.

Script Development Workbench

Shortcut Key Combination	Action
CTRL+N	Open the New dialog box, from which you can create new scripts or datapools.
CTRL+O	Open the Open dialog box, from which you can open various product files.
CTRL+S	Save the active document.
CTRL+P	Print the active document
CTRL+Z	Undo the last action.
CTRL+Y	Redo the last action.
CTRL+X	Cut.
CTRL+C	Copy.
CTRL+V	Paste.

CTRL+F	Find.
CTRL+H	Replace.
CTRL+G	Open the Go to Line dialog box, from which you can go to a specific line number in the active script.
CTRL+F2	Toggle bookmark on/off.
F2	Move to next bookmark.
SHIFT+F2	Move to previous bookmark.
CTRL+SHIFT+F2	Clear all bookmarks.
CTRL+ALT+C	Start a recording.
CTRL+ALT+T	Stop a recording.
CTRL+ALT+R	Restart a recording.
CTRL+ALT+U	Resume a recording.
CTRL+ALT+A	Animate a recording.
CTRL+ALT+S	Step to the next response in an animation.
CTRL+ALT+B	Step back to the previous response in an animation.
F7	Compile the active script.
CTRL+F5	Validate the active script.
CTRL+T	Open the FTP Transfer dialog box, from which you can send a file via FTP.
F1	Open the QALoad online help.
ALT+F4	Close the Script Development Workbench.

Conductor

Shortcut Key Combination	Description
F1	Open the QALoad online help.
CTRL+X	Cut.
CTRL+C	Copy.
CTRL+P	Paste.
ALT+F4	Close the Conductor.

Analyze

Shortcut Key Combination	Description
CTRL+O	Open a timing file.
CTRL+P	Print the active document.
CTRL+C	Copy Summary and Data selections.
CTRL+A	Select all items.
CTRL+U	Unselect all items.
F1	Open QALoad online help.
ALT+F6	Move to the next tab in the Data window.
ALT+F4	Close Analyze.

Player

Shortcut Key Combination	Description
CTRL+N	Clear any current messages from the Player's main window.
CTRL+S	Save a text file of the messages reported by a QALoad Player during a test.
CTRL+P	Print Player output.
CTRL+Z	Undo the last action.
CTRL+X	Cut.
CTRL+C	Copy.
CTRL+V	Paste.
F1	Open QALoad online help.
ALT+F4	Close the Player.

To request a Voluntary Product Accessibility Template (VPAT) or for more information regarding accessibility of QALoad or any other Compuware product, see http://www.compuware.com/accessibility/

Glossary

QALoad glossary

Non-alphabetic

2-tier
3-tier

A

ActiveData
Analyze

B

batch test

C

capture: see recording
.CAP file: see capture file
capture file
checkpoints
checkpoint duration
concurrent users
Conductor
conversion
convert: see conversion
counters

D

datapool

E

EasyScript session
error handling

F

Function Wizard

G

H

I

ICA files
IP spoofing

J

K

L

load testing

M

metrics
middleware
middleware session: see session (middleware)

N

n-tier

O

P

pacing
parameterization: see variablization
performance testing
Player
Player agent
process mode
.PTF file

Q

R

ramp-up script
.REC file

recording
Remote Monitoring
replay
response time
.RIP file

S

Script Development Workbench
service level threshold
session (middleware)
session ID file
sleep factor
sleep
source variable
stress testing
synchronization

T

thread mode
thresholds
.TIM file
timing files
traffic
transaction
transaction cleanup
transaction duration
transaction loop
transaction pacing: see pacing
transaction throughput

U

Universal session

V

validation
variablization
virtual user
Visual Navigator
visual scripting

W

X

Y

Z

Test Execution
Hits (as in, hits per second)
Hits per Second
.CSV File
Rectangular Datapool
QARun
Compiler
Compiler Options
Analyze
Analysis
Conductor
Player
Player Agent
Windows
Unix
Linux
'C'
'C++'
Programming Language
Methodology
Service Level Threshold
Session Duration
Load Test Recording
Debugging
Debug Options
DO_SLEEP
Timing Options
External Data

Binary Files

ASCII Files

Server Analysis Agent

SNMP

Application Expert

ApplicationVantage

Server Vantage

VU

Virtual User Increment

Time Interval

analyze (QALoad)

application under test (AUT)

asynchronous

automated (v. manual)

capture

checkpoint

client

compile

conductor (QALoad)

convert

error

function

KPI-key performance indicator

manual (v. automated)

metrics

reponse time

request

response

script

server

session (.id file)

stateless

synchronous

test plan

testing methodology

workbench (QALoad)

Index

_

_xClone 167

_xResync 168

_xSave 169

A

AddNew 117

AddrByte 505

ADO 1

ADO_Command 15, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

ADO_Command(n)->Cancel 15

ADO_Command(n)->CreateParameter 15

ADO_Command(n)->Execute 18

ADO_Command(n)->GetCommandStream 19

ADO_Command(n)->GetCommandText 19

ADO_Command(n)->GetCommandTimeout 20

ADO_Command(n)->GetCommandType 20

ADO_Command(n)->GetDialect 21

ADO_Command(n)->GetName 21

ADO_Command(n)->GetNamedParameters 22

ADO_Command(n)->GetParameters 23

ADO_Command(n)->GetPrepared 23

ADO_Command(n)->GetProperties 24

ADO_Command(n)->PutActiveConnection 24

ADO_Command(n)->PutCommandText 25

ADO_Command(n)->PutCommandTimeout 26

ADO_Command(n)->PutCommandType 26

ADO_Command(n)->PutDialect 27

ADO_Command(n)->PutName 28

ADO_Command(n)->PutNamedParameters 29

ADO_Command(n)->PutPrepared 29

ADO_Command(n)->PutRefActiveConnection 30

ADO_Connect ... 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 44, 45, 46, 47, 48, 49, 50

ADO_Connect(n)-> GetIsolationLevel 37

ADO_Connect(n)->BeginTrans 30

ADO_Connect(n)->Close 31

ADO_Connect(n)->CommitTrans 32

ADO_Connect(n)->Execute 32

ADO_Connect(n)->GetAttributes 33

ADO_Connect(n)->GetCommandTimeout 34

ADO_Connect(n)->GetConnectionString 34

ADO_Connect(n)->GetConnectionTimeout 35

ADO_Connect(n)->GetCursorLocation 36

ADO_Connect(n)->GetDefaultDatabase 36

ADO_Connect(n)->GetMode 37

ADO_Connect(n)->GetProvider 38

ADO_Connect(n)->GetState 39

ADO_Connect(n)->GetVersion 39

ADO_Connect(n)->Open 40

ADO_Connect(n)->OpenSchema 41

ADO_Connect(n)->PutAttributes 44

ADO_Connect(n)->PutCommandTimeout 44

ADO_Connect(n)->PutConnectionString 45

ADO_Connect(n)->PutConnectionTimeout 46

ADO_Connect(n)->PutCursorLocation 46

ADO_Connect(n)->PutDefaultDatabase 47

ADO_Connect(n)->PutIsolationLevel 48

ADO_Connect(n)->PutMode 49

ADO_Connect(n)->PutProvider 50

ADO_Connect(n)->RollbackTrans 50

ADO_Field ... 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67

ADO_Field(n)->AppendChunk 51

ADO_Field(n)->GetActualSize 52

ADO_Field(n)->GetAttributes 53

ADO_Field(n)->GetChunk 53

ADO_Field(n)->GetDataFormat 54
ADO_Field(n)->GetDefinedSize 55
ADO_Field(n)->GetName 55
ADO_Field(n)->GetNumericScale 56
ADO_Field(n)->GetOriginalValue 57
ADO_Field(n)->GetPrecision 57
ADO_Field(n)->GetProperties 58
ADO_Field(n)->GetStatus 58
ADO_Field(n)->GetType 59
ADO_Field(n)->GetUnderlyingValue 60
ADO_Field(n)->GetValue 60
ADO_Field(n)->PutAttributes 61
ADO_Field(n)->PutDefinedSize 62
ADO_Field(n)->PutNumericScale 63
ADO_Field(n)->PutPrecision 64
ADO_Field(n)->PutRefDataFormat 64
ADO_Field(n)->PutType 65
ADO_Field(n)->PutValue 67
ADO_FieldSet 67, 70, 73, 74, 75, 76, 77
ADO_FieldSet(n)->Append 67
ADO_FieldSet(n)->Append15 70
ADO_FieldSet(n)->CancelUpdate 73
ADO_FieldSet(n)->Delete 74
ADO_FieldSet(n)->GetCount 74
ADO_FieldSet(n)->GetItem 75
ADO_FieldSet(n)->Refresh 76
ADO_FieldSet(n)->Resync 77
ADO_FieldSet(n)->Update 77
ADO_IEnum 79
ADO_IEnum(n)->NextProperty 79
ADO_IEnumField 78
ADO_IEnumField(n)->NextField 78
ADO_IEnumParameter 79
ADO_IEnumParameter(n)->NextParameter 79
ADO_LoadVariant 80
ADO_LoadVariant(n) 80
ADO_Parameter .81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92
ADO_Parameter(n)->AppendChunk 81
ADO_Parameter(n)->GetAttributes 82
ADO_Parameter(n)->GetDirection 82
ADO_Parameter(n)->GetName 83
ADO_Parameter(n)->GetNumericScale 83
ADO_Parameter(n)->GetPrecision 84
ADO_Parameter(n)->GetSize 85
ADO_Parameter(n)->GetValue 85
ADO_Parameter(n)->PutAttributes 86
ADO_Parameter(n)->PutDirection 87
ADO_Parameter(n)->PutName 87
ADO_Parameter(n)->PutNumericScale 88
ADO_Parameter(n)->PutPrecision 89
ADO_Parameter(n)->PutSize 89
ADO_Parameter(n)->PutType 90
ADO_Parameter(n)->PutValue 92
ADO_ParameterSet 93, 94, 95, 96
ADO_ParameterSet(n)->Append 93
ADO_ParameterSet(n)->Delete 93
ADO_ParameterSet(n)->GetCount 94
ADO_ParameterSet(n)->GetItem 94
ADO_ParameterSet(n)->GetNewEnum 95
ADO_ParameterSet(n)->Refresh 96
ADO_Property 96, 97, 98, 100
ADO_Property(n)->GetAttributes 96
ADO_Property(n)->GetName 97
ADO_Property(n)->GetType 97
ADO_Property(n)->GetValue 98
ADO_Property(n)->PutAttributes 99
ADO_Property(n)->PutValue 100
ADO_PropertySet 100, 101, 102
ADO_PropertySet(n)->GetCount 100
ADO_PropertySet(n)->GetItem 101
ADO_PropertySet(n)->GetNewEnum 101
ADO_PropertySet(n)->Refresh 102

ADO_Record.... 102, 103, 105, 106, 107, 108, 109, 110, 111, 113, 114, 115, 116
ADO_Record(n)->Cancel 102
ADO_Record(n)->Close 103
ADO_Record(n)->CopyRecord 103
ADO_Record(n)->DeleteRecord 105
ADO_Record(n)->GetActiveConnection 105
ADO_Record(n)->GetChildren 106
ADO_Record(n)->GetFields 106
ADO_Record(n)->GetMode 107
ADO_Record(n)->GetParentURL 108
ADO_Record(n)->GetRecordType 108
ADO_Record(n)->GetSource 109
ADO_Record(n)->GetState 109
ADO_Record(n)->MoveRecord 110
ADO_Record(n)->Open 111
ADO_Record(n)->PutActiveConnection 113
ADO_Record(n)->PutMode 114
ADO_Record(n)->PutRefActiveConnection 115
ADO_Record(n)->PutSource 116
ADO_Recordset 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 166, 167, 168, 169
ADO_Recordset(n)->_xClone 167
ADO_Recordset(n)->_xResync 168
ADO_Recordset(n)->_xSave 169
ADO_Recordset(n)->AddNew 117
ADO_Recordset(n)->Cancel 117
ADO_Recordset(n)->CancelBatch 118
ADO_Recordset(n)->CancelUpdate 119
ADO_Recordset(n)->Clone 119
ADO_Recordset(n)->Close 120
ADO_Recordset(n)->CompareBookmarks 121
ADO_Recordset(n)->Delete 122
ADO_Recordset(n)->Find 122
ADO_Recordset(n)->GetAbsolutePage 123
ADO_Recordset(n)->GetAbsolutePosition 124
ADO_Recordset(n)->GetActiveCommand 125
ADO_Recordset(n)->GetActiveConnection 125
ADO_Recordset(n)->GetBOF 126
ADO_Recordset(n)->GetBookmark 126
ADO_Recordset(n)->GetCacheSize 127
ADO_Recordset(n)->GetCollect 128
ADO_Recordset(n)->GetCursorLocation 128
ADO_Recordset(n)->GetCursorType 129
ADO_Recordset(n)->GetDataMember 130
ADO_Recordset(n)->GetDataSource 130
ADO_Recordset(n)->GetEditMode 131
ADO_Recordset(n)->GetEOF 131
ADO_Recordset(n)->GetFields 132
ADO_Recordset(n)->GetFilter 133
ADO_Recordset(n)->GetIndex 133
ADO_Recordset(n)->GetLockType 134
ADO_Recordset(n)->GetMarshalOptions 134
ADO_Recordset(n)->GetMaxRecords 135
ADO_Recordset(n)->GetPageCount 135
ADO_Recordset(n)->GetPageSize 136
ADO_Recordset(n)->GetProperties 137
ADO_Recordset(n)->GetRecordCount 137
ADO_Recordset(n)->GetRows 138
ADO_Recordset(n)->GetSort 138
ADO_Recordset(n)->GetSource 139
ADO_Recordset(n)->GetState 139
ADO_Recordset(n)->GetStatus 140
ADO_Recordset(n)->GetStayInSync 140
ADO_Recordset(n)->GetString 141
ADO_Recordset(n)->Move 142
ADO_Recordset(n)->MoveFirst 143
ADO_Recordset(n)->MoveLast 143
ADO_Recordset(n)->MoveNext 144
ADO_Recordset(n)->MovePrevious 145
ADO_Recordset(n)->NextRecordset 145
ADO_Recordset(n)->Open 146

ADO_Recordset(n)->PutAbsolutePage 147
ADO_Recordset(n)->PutAbsolutePosition 148
ADO_Recordset(n)->PutActiveConnection 149
ADO_Recordset(n)->PutBookmark 150
ADO_Recordset(n)->PutCacheSize..... 150
ADO_Recordset(n)->PutCollect 151
ADO_Recordset(n)->PutCursorLocation 152
ADO_Recordset(n)->PutCursorType 152
ADO_Recordset(n)->PutDataMember 153
ADO_Recordset(n)->PutFilter..... 154
ADO_Recordset(n)->PutIndex..... 154
ADO_Recordset(n)->PutLockType 155
ADO_Recordset(n)->PutMarshalOptions..... 156
ADO_Recordset(n)->PutMaxRecords 156
ADO_Recordset(n)->PutPageSize 157
ADO_Recordset(n)->PutRefActiveConnection 158
ADO_Recordset(n)->PutRefDataSource 158
ADO_Recordset(n)->PutRefSource..... 159
ADO_Recordset(n)->PutSort 159
ADO_Recordset(n)->PutSource 160
ADO_Recordset(n)->PutStayInSync..... 161
ADO_Recordset(n)->ReQuery 161
ADO_Recordset(n)->Resync..... 162
ADO_Recordset(n)->Save..... 163
ADO_Recordset(n)->Seek 164
ADO_Recordset(n)->Supports..... 164
ADO_Recordset(n)->Update..... 166
ADO_Recordset(n)->UpdateBatch 166
ADO_Stream ... 169, 170, 171, 172, 173, 174, 175, 176, 177, 179, 180, 181, 182, 183, 184, 185, 186
ADO_Stream(n)->Cancel 169
ADO_Stream(n)->Close..... 170
ADO_Stream(n)->CopyTo..... 170
ADO_Stream(n)->Flush 171
ADO_Stream(n)->GetCharset 172
ADO_Stream(n)->GetEOS..... 172
ADO_Stream(n)->GetLineSeparator..... 173
ADO_Stream(n)->GetMode 174
ADO_Stream(n)->GetPosition 174
ADO_Stream(n)->GetSize 175
ADO_Stream(n)->GetState.....175
ADO_Stream(n)->GetType.....176
ADO_Stream(n)->LoadFromFile 177
ADO_Stream(n)->PutCharset.....179
ADO_Stream(n)->PutLineSeparator.....179
ADO_Stream(n)->PutMode.....180
ADO_Stream(n)->PutPosition.....181
ADO_Stream(n)->PutType.....182
ADO_Stream(n)->Read.....182
ADO_Stream(n)->ReadText 183
ADO_Stream(n)->SaveToFile 184
ADO_Stream(n)->SetEOS.....185
ADO_Stream(n)->SkipLine 185
ADO_Stream(n)->Write 186
ADO_Stream(n)->WriteText 186
ADOStream 177
ADOStream(n)->Open 177
Append 67, 93
Append15 70
AppendChunk 51, 81
Attach 541

B

BEGIN_CHECKPOINT 408
BEGIN_TRANSACTION 408
BEGIN_UENTITY 476
BeginBlock 189
BeginCheckpoint 408
BeginTrans.....30

C

Cancel 15, 102, 117, 169
CancelBatch 118
CancelUpdate 73, 119
Citrix command index 187
CitrixInit.....190

CitrixUninit 190
Clear 541
Click, Citrix command 191
Click_On 543
Clone 119
Close 31, 103, 120, 170
CLOSE_ALL_DATA_POOLS 409
CLOSE_DATA_POOL 409
CommitTrans 32
CompareBookmarks 121
Connect, Citrix command 192
CopyRecord 103
CopyTo 170
COUNTER_VALUE 410
CreateParameter 15
CtxClick 191
CtxConnect 192
CtxDisconnect 193
CtxDoubleClick 194
CtxKeyDown 195
CtxKeyUp 195
CtxMouseDown 196
CtxMouseMove 197
CtxMouseUp 198
CtxPing 199
CtxPoint 199
CtxScreenEventExists 199
CtxSetApplication 200
CtxSetCitrixPort 201
CtxSetConnectTimeout 201
CtxSetDisconnectTimeout 202
CtxSetEnableCounters 202
CtxSetEnableWildcardMatching 203
CtxSetICAFile 204
CtxSetLoginInfo 204
CtxSetPingTimeout 205
CtxSetWaitPointTimeout 205
CtxSetWindowMatchTitle 206
CtxSetWindowRetries 206
CtxSetWindowTimeout 207
CtxSetWindowVerification 207
CtxType 208
CtxTypeChar 208
CtxTypeVK 209
CtxWaitForCaptionChange 210
CtxWaitForScreenUpdate 210
CtxWaitForWindowActivate 211
CtxWaitForWindowCreate 212
CtxWaitforWindowDestroy 212
CtxWaitForWindowLgIconChange 213
CtxWaitForWindowMinimize 213
CtxWaitForWindowMove 214
CtxWaitForWindowResize 214
CtxWaitForWindowSmIconChange 215
CtxWaitforWindowStyleChange 216
CtxWindowEventExists 216
customer support 739

D

datapools
substituting a string 437
DATE_TIME 410
DefaultCheckpointsOn 411
DEFINE_COUNTER 412
DEFINE_TRANS_TYPE 413
Delete 74, 93, 122
DeleteRecord 105
DisableStatisticsRP 544
Disconnect, Citrix command 193
DO_AbortOnError 413
DO_AddHeader 544
DO_AdditionalSubRequest 545
DO_AllowTrafficFrom 546
DO_AttachFile 546
DO_autocommitoff 271

DO_autocommiton 271
DO_AutomaticSubRequests 547
DO_BasicAuthorization 549
DO_binddate 272
DO_BindForUpdateRowID 273
DO_bindnull 274
DO_bindstring 274
DO_BindV 275
DO_BlankOutOfRangeData 550
DO_BlockTrafficFrom 550
DO_Cache 551
DO_cleanup 276
DO_Clear 551
DO_ClearCache 553
DO_ClearDNSCache 554
DO_ClearJavascript 554
DO_commit 277
DO_DynamicCookieHandling 555
DO_DynamicRedirectHandling 556
DO_EnableJavascript 558
DO_EncodeString 559
Do_ExtractString 414
DO_FetchIters 277
DO_free_data 266
DO_FreeHttp 559
DO_freeitem 266
DO_FreeODBC 222
DO_get_select_variable 279
DO_GetAnchorByNumber 560
DO_GetAnchorCount 561
DO_GetAnchorHREF 561
DO_GetAnchorHREFEx 562
DO_GetAnchorHREFn 564
DO_GetClientMapHREF 565
DO_GetCookie 566
DO_GetCookieFromReplyEx 567
DO_GetFormActionStatement 569
DO_GetFormValueByName 570
DO_GetHeaderFromReply 571
DO_GetLastHttpError 572
DO_GetOutputData 267
DO_GetRedirectedURL 572
DO_GetReplyBuffer 573
DO_GetSelectData 278
DO_GetUniqueString 574
DO_GetUniqueStringEx 575
DO_Http 575
DO_HttpCleanup 576
DO_Https 468
DO_HTTPVersion 577
DO_init_alen 280
DO_init_data 281
DO_init_indp 282
DO_InitHttp 577
DO_initODBC 223
DO_IPSpoofEnable 578
DO_LoadMem 223
DO_Logfile_URB 477
DO_makedate 268
DO_MSLEEP 415
DO_NTLMAuthorization 579
DO_OCI8BindDate 296
DO_OCI8BindNull 297
DO_OCI8BindString 298
DO_OCI8GetSelectData 299
DO_OCI8InitAlen 301
DO_OCI8InitIndp 300
DO_OCIAttrSet 302
DO_OCIBind 303
DO_OCICommit 305
DO_OCIDefine 306
DO_OCIDescriptorAlloc 307
DO_OCIDescriptorFree 308
DO_OCIEnvFreeAll 308

DO_OCIEnvInit 309
DO_OCIExecute 310
DO_OCIHandleAlloc 310
DO_OCIHandleFree 311
DO_OCIInitialize 312
DO_OCILdaToSvcCtx 313
DO_OCILobRead 313
DO_OCILobWrite 314
DO_OCILogoff 315
DO_OCILogoffEx 316
DO_OCILogon 317
DO_OCIProcessSelectList 318
DO_OCIProcessSelectList_EX 318
DO_OCIRollback 319
DO_OCIServerAttach 320
DO_OCIServerDetach 321
DO_OCISessionBegin 322
DO_OCISessionEnd 323
DO_OCIStmtExecute 324
DO_OCIStmtPrepare 324
DO_OCIStmtPrepare_EX 325
DO_OCISvcCtxToLda 326
DO_OCITransCommit 327
DO_OCITransRollback 327
DO_oclose 283
DO_oexec 283
DO_olog 284
DO_ologof 285
DO_oopen 285
DO_oopt 286
DO_oparse 286
DO_process_select_list 287
DO_ProxyAuthorization 580
DO_ProxyExceptions 581
DO_rollback 288
Do_SAPCheckScreen 439
Do_SAPCheckStatus 440
Do_SAPCheckTitle 440
Do_SAPClearMessages 441
Do_SAPDumpEvent 441
Do_SAPExtractString 442
Do_SAPFrontEnd 442
Do_SAPFullMenu 443
Do_SAPGetControlValue 444
Do_SAPGetIt_Event 444
Do_SAPGetScreenTitle 445
Do_SAPGetStatusMsg 445
Do_SAPInit 446
Do_SAPLogging 446
Do_SAPLogin 447
Do_SAPLogoff 447
Do_SAPSendDblClick 448
Do_SAPSendEvent 448
Do_SAPSendMenu 449
Do_SAPSendOKCode 449
Do_SAPSendPFKey 449
Do_SAPSendReturn 450
Do_SAPSetCheckScreenWildcard 450
Do_SAPSetCtrlValue 451
Do_SAPSetCursor 451
DO_SaveReplyType 582
DO_ScalarBindA 288
DO_SetAssumedContentType 582
DO_SetBaudRate 583
DO_SetBaudRateEx 584
DO_SetCookie 585
DO_SetCookieEx 586
DO_SetJavascriptCleanupThreshold 587
DO_SetMaxBrowserThreads 587
DO_SetMaximumRetries 588
DO_SetRefreshTimeout 588
DO_SetRetryWait 589
DO_SetSSLConnectString 469
DO_SetTimeout 589

DO_SetTransactionCleanup 416
DO_SetTransactionStart 416
DO_SetValue 417
DO_SLEEP 418
DO_SoftClose 289
DO_SQLAllocConnect 224
DO_SQLAllocHandle 225
DO_SQLAllocStmt 225
DO_SQLBindCol 226
DO_SQLBindParameter 228
DO_SQLCancel 230
DO_SQLCloseCursor 231
DO_SQLColAttribute 231
DO_SQLColumns 232
DO_SQLConnect 233
DO_SQLCopyDesc 234
DO_SQLDescribeCol 235
DO_SQLDisconnect 236
DO_SQLDriverConnect 236
DO_SQLEndTran 237
DO_SQLExecDirect 237
DO_SQLExecute 238
DO_SQLFetch 238
DO_SQLFreeConnect 239
DO_SQLFreeHandle 240
DO_SQLFreeStmt 240
DO_SQLGetCursorName 241
DO_SQLGetData 242
DO_SQLGetDescField 243
DO_SQLGetDescRec 244
DO_SQLGetEnvAttr 244
DO_SQLGetTypeInfo 245
DO_SQLNumResultCols 247
DO_SQLParamData 247
DO_SQLPrepare 248
DO_SQLPutData 248
DO_SQLRetrieveParamValue 249
DO_SQLRowCount 250
DO_SQLSetConnectAttr 250
DO_SQLSetConnectOption 251
DO_SQLSetCursorName 253
DO_SQLSetDescField 253
DO_SQLSetDescRec 254
DO_SQLSetEnvAttr 255
DO_SQLSetPos 256
DO_SQLSetStmtAttr 257
DO_SQLSetStmtOption 257
DO_SQLSpecialColumns 260
DO_SQLStatistics 262
DO_SQLTables 263
DO_SQLTransact 264
DO_SSLReuseSession 470
DO_SSLUseCipher 471
DO_SSLUseClientCert 472
DO_SSLUseClientCertPass 472
DO_SSLUseProxy 473
DO_strdup 269
DO_substr 264
DO_URB_AsciiToHex 477
DO_URB_Init 478
DO_URB_setoprretry 478
DO_URB_ubin2uf 478
DO_URB_udbl2uf 479
DO_URB_uecreate 479
DO_URB_uedelete 480
DO_URB_uentcreo 481
DO_URB_uentoccs 482
DO_URB_uentseto 482
DO_URB_ufreeh 483
DO_URB_uinstdel 483
DO_URB_uinstnew 484
DO_URB_uinstopr 485
DO_URB_ulist2uf 485
DO_URB_ulistdel 486

DO_URB_ulistfree 487
DO_URB_ulistget 487
DO_URB_ulistnew 489
DO_URB_ulistput 489
DO_URB_ulistputlist 490
DO_URB_ulistputx 491
DO_URB_ulong2uf 491
DO_URB_unifree 492
DO_URB_uniname 492
DO_URB_uopract 493
DO_URB_uoprprms 494
DO_URB_uprmdir 494
DO_URB_uprmgeth 495
DO_URB_uprmtype 496
DO_URB_ustr2uf 496
DO_URB_uuf2bin 497
DO_URB_uuf2dbl 498
DO_URB_uuf2list 498
DO_URB_uuf2long 499
DO_URB_uuf2str 500
DO_UseEntityList 590
DO_UseNumericReferenceList 590
DO_UsePersistentConnections 591
DO_UseProxy 591
DO_UseProxyAutomaticConfiguration 592
DO_VerifyDocTitle 593
DO_WSK_Accept 505
DO_WSK_Bind 506
DO_WSK_Closesocket 506
DO_WSK_Connect 507
DO_WSK_Expect 508
DO_WSK_ExpectAny 508
DO_WSK_ExpectAnyExpr 509
DO_WSK_ExpectExpr 509
DO_WSK_GetSocket 510
DO_WSK_Getsockname 511
DO_WSK_HexDecode 511
DO_WSK_Init 512
DO_WSK_Ioctlsocket 512
DO_WSK_IsReadable 513
DO_WSK_IsWritable 513
DO_WSK_IsWriteable 513
DO_WSK_Listen 514
DO_WSK_Quiet 514
DO_WSK_Read 515
DO_WSK_Recv 516
DO_WSK_Recvfrom 516
DO_WSK_Reorder 517
DO_WSK_Select 518
DO_WSK_Send 519
DO_WSK_SendAll 519
DO_WSK_Sendto 520
DO_WSK_SetsockOpt 520
DO_WSK_Shutdown 521
DO_WSK_Socket 522
DO_WSK_Write 523
DoubleClick, Citrix command 194
DownloadMediaFromASX 593
DownloadMediaRP 594
DownloadMediaWMP 595
DownloadWindowsMedia 595

E

EnableStatisticsRP 596
END_CHECKPOINT 419
END_TRANSACTION 419
END_UENTITY 501
EndBlock 218
EndCheckpoint 419
EscapeStr 523
Execute 18, 32
EXIT 420

F

Fill_In 597
Find, ADO command 122

Flush 171

G

General Oracle 266
Get 597
GET_ABSOLUTE_VUNUM 420
Get_Data 421
GET_DATA_FIELD 421
GET_DATAPOOLS_DIR 422
GET_HOME_DIR 422
GET_LOGFILES_DIR 423
GET_RELATIVE_VUNUM 423
GET_SCRIPTS_DIR 424
GET_TIMINGFILES_DIR 424
GetAbsolutePage 123
GetAbsolutePosition 124
GetActiveCommand 125
GetActiveConnection 105, 125
GetActualSize 52
GetAttributes 33, 53, 82, 96
GetBindColumnData 265
GetBOF 126
GetBookmark 126
GetCacheSize 127
GetCharset 172
GetChildren 106
GetChunk 53
GetCollect 128
GetCommandStream 19
GetCommandText 19
GetCommandTimeout 20, 34
GetCommandType 20
GetConnectionString 34
GetConnectionTimeout 35
GetCount 74, 94, 100
GetCursorLocation 36, 128
GetCursorType 129
GetDataFormat 54
GetDataMember 130
GetDataSource 130
GetDefaultDatabase 36
GetDefinedSize 55
GetDialect 21
GetDirection 82
GetEditMode 131
GetEOF 131
GetEOS 172
GetFields 106, 132
GetFilter 133
GetIndex 133
GetIsolationLevel 37
GetItem 75, 94, 101
GetLineSeparator 173
GetLocalAddr 524
GetLocalPort 525
GetLockType 134
GetMarshalOptions 134
GetMaxRecords 135
GetMode 37, 107, 174
GetName 21, 55, 83, 97
GetNamedParameters 22
GetNewEnum 95, 101
GetNumericScale 56, 83
GetOriginalValue 57
GetPageCount 135
GetPageSize 136
GetParameters 23
GetParentURL 108
GetPosition 174
GetPrecision 57, 84
GetPrepared 23
GetProperties 24, 58, 137
GetProvider 38
GetRecordCount 137
GetRecordType 108

GetRemoteAddr 525
GetRemotePort 526
GetRows 138
GetSize 85, 175
GetSort 138
GetSource 109, 139
GetState 39, 109, 139, 175
GetStatus 58, 140
GetStayInSync 140
GetString 141
GetType 59, 97, 176
GetUnderlyingValue 60
GetValue 60, 85, 98
GetVersion 39
glossary 746

Group

ADO

General Oracle 266, 267, 268, 269
ODBC 222, 223, 228
ODBC/DB2 223, 224, 225, 226, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 247, 248, 249, 250, 251, 253, 254, 255, 256, 257, 260, 262, 263, 264
Oracle Forms Server 403, 404
Oracle OCI version 7 271, 272, 273, 274, 275, 276, 277, 278, 279, 282, 283, 284, 285, 286, 287, 288, 289
Oracle OCI version 8 296, 297, 298, 299, 300, 301, 302, 303, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327
QALoad 408, 409, 410, 411, 412, 413, 416, 417, 418, 419, 420, 421, 422, 423, 424, 426, 427, 429, 431, 434, 435, 436, 437
SAP ... 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451
SSL 468, 469, 470, 471, 472, 473
UNIFACE .. 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500
WinSock ... 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534
WWW 541, 543, 544, 545, 546, 547, 549, 550, 551, 553, 554, 555, 556, 558, 559, 560, 561, 562, 564, 565, 566, 567, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 601, 602, 603, 604, 605

ADO 15, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 70, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 105, 106, 107, 108, 109, 110, 111, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 179, 180, 181, 182, 183, 184, 185, 186

H

HiByte 526

K

KeyDown 195
KeyUp 195

L

LoadFromFile 177
LoByte 527
Log 527
LOG_ERROR 425

M

MouseDown 196
MouseMove 197
MouseUp 198
Move 142
MoveFirst 143
MoveLast 143

MoveNext 144
MovePrevious..... 145
MoveRecord 110
MyByteOrder..... 527
N
Navigate_To 598
NextField..... 78
NextParameter 79
NextProperty..... 79
NextRecordset 145
O
OctalToChar 426
ODBC/DB2..... 218
ofsActivateListItem 334
ofsActivateTreeItem 335
ofsActivateWindow..... 336
ofsClickButton 336
ofsClickTextFieldItem 337
ofsClosePopList 338
ofsCloseWindow..... 339
ofsCollapseTreeItem 340
ofsColorAdd..... 341
ofsConnectToSocket 342
ofsDeActivateWindow 342
ofsDefineTreeNode 343
ofsDefineTreeNodeOffset..... 344
ofsDeIconifyWindow..... 345
ofsDeSelectItem 346
ofsDeselectTreeEvent 347
ofsEdit 348
ofsExpandTreeItem 348
ofsFindLOVVAlue 349
ofsFocus..... 350
ofsGetServerData 351
ofsHideWindow 352
ofsHTTPConnectToFormsServlet..... 353
ofsHTTPConnectToListenerServlet..... 353
ofsHTTPDisconnect 354
ofsHTTPInitialFormsConnect.....354
ofsHTTPSDoSSLHandshake 354
ofsHTTPSetHdrProperty.....355
ofsHTTPSetListenerServletParms.....355
ofsIconifyWindow 356
ofsIndexKey 357
ofsIndexSKey 357
ofsInitSessionCmdLine.....358
ofsInitSessionTimeZone 359
ofsListItemValue.....360
ofsLoadValue 360
ofsLOVRequestRow 361
ofsLOVSelection 362
ofsMenuParamDlgOK.....363
ofsOpenWindow 363
ofsRemoveFocus 364
ofsScroll 365
ofsScrollSize 366
ofsSelectItem 366
ofsSelectMenuItem 367
ofsSelectTreeEvent 368
ofsSendRecv 369
ofsServerSideDisconnect.....369
ofsSetColorDepth 370
ofsSetCursorPosition.....370
ofsSetDisplaySize 371
ofsSetErrorDialogTitle.....372
ofsSetExpectedServerMsg.....373
ofsSetFontName.....373
ofsSetFontSize.....374
ofsSetFontStyle 375
ofsSetFontWeight 375
ofsSetICXTicket 376
ofsSetInitialVersion 377
ofsSetJavaContainerArgName 378
ofsSetJavaContainerArgValue.....379

ofsSetJavaContainerEvent 380
ofsSetLogonDatabase 380
ofsSetLogonPassWord 381
ofsSetLogonUserName 382
ofsSetNoRequiredVAList 382
ofsSetPropertyBoolean 383
ofsSetPropertyByte 384
ofsSetPropertyByteArray 385
ofsSetPropertyCharacter 385
ofsSetPropertyDate 386
ofsSetPropertyFloat 387
ofsSetPropertyInteger 388
ofsSetPropertyPoint 388
ofsSetPropertyRectangle 389
ofsSetPropertyString 390
ofsSetPropertyStringArray 391
ofsSetPropertyVoid 391
ofsSetRequiredVAList 392
ofsSetRunOptions 393
ofsSetScaleInfo 394
ofsSetScreenResolution 395
ofsSetSelection 396
ofsSetServerFailedMsg 397
ofsSetServletMode 397
ofsSetValue 399
ofsSetWindowLocation 398
ofsSetWindowSize 399
ofsShowWindow 400
ofsSocketDisconnect 401
ofsStartSubMessage 401
ofsTabControlTopPage 402
ofsUnSetPropertyBoolean 403
Open 40, 111, 146, 177
OPEN_DATA_POOL 427
OpenSchema 41
Oracle Forms Server 328
Oracle OCI Version 7 269
Oracle OCI Version 8 290
OracleAppsLogin 403

P

password-protected directory 549
Ping 199
PlayMedia 599
Point 199
PopupHelp 404
Post_To 599
PutAbsolutePage 147
PutAbsolutePosition 148
PutActiveConnection 24, 113, 149
PutAttributes 44, 61, 86
PutBookmark 150
PutCacheSize 150
PutCharset 179
PutCollect 151
PutCommandText 25
PutCommandTimeout 26, 44
PutCommandType 26
PutConnectionString 45
PutConnectionTimeout 46
PutCursorLocation 46, 152
PutCursorType 152
PutDataMember 153
PutDefaultDatabase 47
PutDefinedSize 62
PutDialect 27
PutDirection 87
PutFilter 154
PutIndex 154
PutIsolationLevel 48
PutLineSeparator 179
PutLockType 155
PutMarshalOptions 156
PutMaxRecords 156
PutMode 49, 114, 180

PutName 28, 87
PutNamedParameters..... 29
PutNumericScale..... 63, 88
PutPageSize 157
PutPosition..... 181
PutPrecision 64, 89
PutPrepared..... 29
PutProvider 50
PutRefActiveConnection 30, 115, 158
PutRefDataSource..... 158
PutRefSource..... 159
PutSize..... 89
PutSort 159
PutSource..... 116, 160
PutStayInSync..... 161
PutType..... 65, 90, 182
PutValue..... 67, 92, 100

Q

QALoad 405

R

RandNumString..... 600
RANDOM_NUMBER..... 427
RANDOM_STRING..... 428
Read 182
READ_DATA_RECORD 429
ReadText 183
Refresh 76, 96, 102
Region 600
ReQuery 161
Response 528
ResponseLength 529
RESTART_TRANSACTION_BOTTOM..... 601
RESTART_TRANSACTION_TOP..... 601
restarting transactions
DO_SetTransactionCleanup 416
DO_SetTransactionStart 416
Resync..... 77, 162
RND_DELAY 429
RND_DELAY_RANGE 430
RollbackTrans 50
RR__FailedMsg.....430
RR__GetDebugFlag 431

S

SAP command index, Version 6.2.....452
SAP command index, Versions 4.x.....437
SAPGuiApplication 453
SAPGuiCheckScreen 454
SAPGuiCheckStatusbar 455
SAPGuiCmd0.....456
SAPGuiCmd1.....456
SAPGuiCmd1Coll 457
SAPGuiCmd1Elmnt 458
SAPGuiCmd1Sub 459
SAPGuiCmd1Sub1.....459
SAPGuiCmd2.....460
SAPGuiCmd3.....461
SAPGuiConnect 462
SAPGuiCreateColl.....462
SAPGuiDestroyColl.....463
SAPGuiPropIdStr.....464
SAPGuiPropIdStrExists.....464
SAPGuiPropIdStrExistsEnd 465
SAPGuiSessionInfo 465
SAPGuiSetCheckScreenWildcard.....466
SAPGuiVerCheckStr 467
Save.....163
SaveToFile.....184
ScanExpr 529
ScanFloat.....530
ScanInt 530
ScanLenString.....531
ScanRewind 532
ScanSkip.....532
ScanString.....533

SCRIPT_MESSAGE 433
Set 602
SET_ABORT_FUNCTION 434
SetApplication 200
SetCitrixPort 201
SetConnectTimeout 201
SetDisconnectTimeout 202
SetDomainLoginInfo 193
SetEnableCounters 202
SetEnableWildcardMatching 203
SetEOS 185
SetICAFile 204
SetLoginInfo 204
SetPingTimeout 205
SetTimeout 533
SetWaitPointTimeout 205
SetWindowMatchName 206
SetWindowMatchTitle 206
SetWindowRetries 206
SetWindowTimeout 207
SetWindowVerification 207
ShowMediaRP 603
SkipExpr 534
SkipLine 185
SLEEP 435
SSL 468
Supports 164
SYNCH 436
SYNCHRONIZE 436

T

technical support iii
technical support 739
transmitting files to Technical Support 740
Type 208
TypeChar 208
TypeVK 209

U

UFIELD 501
UnEscapeStr 534
Update 77, 166
UpdateBatch 166

V

VARDATA 437
Verify 604

W

WaitForCaptionChange 210
WaitForScreenUpdate 210
WaitForWindowActivate 211
WaitForWindowCreate 212
WaitForWindowDestroy 212
WaitForWindowMinimize 213
WaitForWindowMove 214
WaitForWindowResize 214
WaitForWindowStyleChange 216
Winsock 502
Write 186
WriteText 186
WWW 535
WWW_FATAL_ERROR 604

X

X_Coord 605
XmlRequest 605

Y

Y_Coord 606

Z

ZipFile Wizard 740
zipping 740