Oracle Objects for OLE

Copyright Other Information Sources

For Help on Help, Press F1

Modification History

Version 1.0.42.0 December 1994

Finalized Oracle Objects for OLE

Version 1.0.53.0 (Internal) March 27, 1995

Added Modification History Topic
Fixed references to COUNT property of the OraParameters Collection
Modified Troubleshooting Topic and added section about duplicate DLLs
Added Copyright Topic
Removed Add/Remove methods from OraParameter Object since these are only valid for the OraParameters Collection
Fixed typos in the OraSession Remarks section
Fixed underline typo at the end of the Transactions topic
Fixed typo in the Redistributable file topic
Changed reference of method to property in the RecordCount property
Fixed GetChunk Link in See Also of Error Handling Topic
Fixed all See Also/Properties/Methods in OLESERV.DOC to be consistent as far as not repeating the words property/method, etc.
Fixed alpha ordering of See Also in FieldSize Method
Fixed alpha ordering of See Also in Move Methods
Changed CreateNameSession to CreateNamedSession
Added code to the CreateNamedSession Example
Changed code samples to use DbOpenDatabase instead of OpenDatabase
Added See Also to the following Data Control properties: BackColor, DragIcon, DragMode, Enabled, FontBold, FontItalics, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Index, Left, MousePointer, Top, Visible, Width
Added See Also to the following Data Control Methods: Drag, Move
Added See Also to the following Data Control Events: DragDrop, DragOver, Error, MouseDown, MouseMove, MouseUp, Reposition, Validate
Added extra information to the RecordCount property about nocache
Added ORADYN_NOMOVEFIRST option to CreateDynaset
Fixed ConnectSession & Example, CreateNamedSession & Example and CreateSession about no cross application session sharing
Added version numbers for C++ compilers to the Requirements topic.
Updated Copyright topic.
Added the return value to ExecuteSQL
Added some clarification to the Data Aware Controls section
Updated the "OO4O and VB" section

Version 1.0.55.0 April 1995

Added new values to the Type property and rearranged the table
Added more information to Data Aware Controls
Added Standard/Pro to Requirements for VB3
Added information to AddNew/Edit/Delete about canceling each other
Removed NoMoveFirst option of CreateDynaset
Rephrased the AddNew/Delete/Edit canceling each other sentences in each of those topics
Updated Add, Remove, and ExecuteSQL methods, and ServerType property examples to show how a stored function would be called.

Copyright

Oracle Objects for OLE, Version 1.0

Release 1.0.55.0

Primary Authors: Keith Majkut, Kevin Whitley
Contributors: Jia-Der Day, Geraldine Kuo, Diana Lorentz, Paul Richard, Bill Sisson

Other Information Sources

The following Oracle publications contain more information about various topics mentioned here:

- *Oracle7 Server Application Developer's Guide*
- *Oracle7 Server Concepts Manual*
- *Oracle7 Server SQL Language Quick Reference*
- *Oracle7 Server SQL Language Reference Manual*
- *PL/SQL User's Guide and Reference*
- *PL/SQL V2.1 and Precompiler's V1.6 Addendum*
- *Oracle7 Server Documentation Addendum*

Requirements

See Also

Design Time

- The *Oracle Object Server* requires an application that supports OLE Automation, such as Visual Basic 3.0 (Standard or Professional), Excel 5.0, or Access 2.0.
- The *Oracle Data Control* requires Visual Basic 3.0 (Standard or Professional).
- The *Oracle Objects for OLE C++ Class Library* requires either Microsoft Visual C++ Version 1.5 (16 bit) or Borland C++ Version 4.0/4.5 (16 bit). Neither the OLE SDK nor OLE development knowledge is necessary.

Run Time

- Windows 3.1 or an equivalent environment capable of running 16 bit Windows applications such as Windows NT (using WOW) or OS/2 (using Win-OS/2).
- A local or remote Oracle7 database.
- Oracle SQL*Net Version 1.x or 2.x if connecting to a remote Oracle7 database.
- Microsoft OLE 2.0.1 or greater run time files (included with the Oracle Objects for OLE installation).

See Also

Oracle Data Control
Oracle Object Server
Redistributable Files

Overview

See Also

Oracle Objects for OLE

Oracle Objects for OLE is a collection of programmable objects that simplifies the development of applications designed to communicate with an Oracle7 database. Oracle Objects for OLE is particularly well suited for any programming environment that supports Visual Basic custom controls (VBX) or OLE Automation.   Oracle Objects for OLE consists of three principle components: the *Oracle Object Server*, the *Oracle Data Control*, and the *Oracle Objects for OLE C++ Class Library.*

The Oracle Object Server

The *Oracle Object Server* is an OLE In Process server that supports a collection of programmable objects for Oracle7 databases running either locally or remotely.   An OLE In Process server is a special kind of OLE server, running in a Windows DLL, that supports the OLE Automation interface. An OLE In Process server has no user interface and is not embeddable.   You can access the *Oracle Object Server* through the *Oracle Data Control,* through any application that supports OLE Automation (such as in Visual Basic for Applications, in applications such as Microsoft Excel Version 5.0 and Access 2.0), and through the *Oracle Objects for OLE C++ Class Library.*

The Oracle Data Control

The *Oracle Data Control* is a Visual Basic custom control for use with development tools that support custom controls.   The Oracle Data Control is compatible with the Microsoft data control included with Visual Basic.   If you are familiar with that data control, learning to use the *Oracle Data Control* is quick and easy.

The Oracle Objects for OLE C++ Class Library

The *Oracle Objects for OLE C++ Class Library* is a collection of C++ classes that provide programmatic access to the *Oracle Object Server*.   Although the class library is implemented using OLE Automation, neither the OLE development kit nor any OLE development knowledge is necessary to use it. In addition to the object classes, the class library provides a bound class, which allows controls such as text and list boxes to be linked directly to a field of a dynaset (columns in the database).   The bound class supports late, runtime binding, as is available in Visual Basic.   The *Oracle Objects for OLE C++ Class Library* is supported for Microsoft Visual C++ and (for the bound class) the Microsoft Foundation Classes as well as Borland C++ and (for the bound class) the Object Windows Library.

Available Objects

- OraClient
  An **OraClient** object defines a workstation domain and manages collections of **OraSession** objects.
- OraSession
  An **OraSession** object manages collections of **OraDatabase** objects and provides connection sharing and transactional control.
- OraConnection
  An **OraConnection** object represents a single connection to an Oracle database.
- OraDatabase
  An **OraDatabase** object represents a single virtual login to an Oracle database.

- OraParameter

  An **OraParameter** object represents a bind variable in a SQL statement or PL/SQL block.
- OraDynaset

  An **OraDynaset** object represents the rows and columns returned from a SQL SELECT statement.
- OraField

  An **OraField** object represents a single column within a row of an **OraDynaset** object.

In addition, there exist objects that represent collections of **OraSession**, **OraConnection**, **OraParameter**, and **OraField** objects.

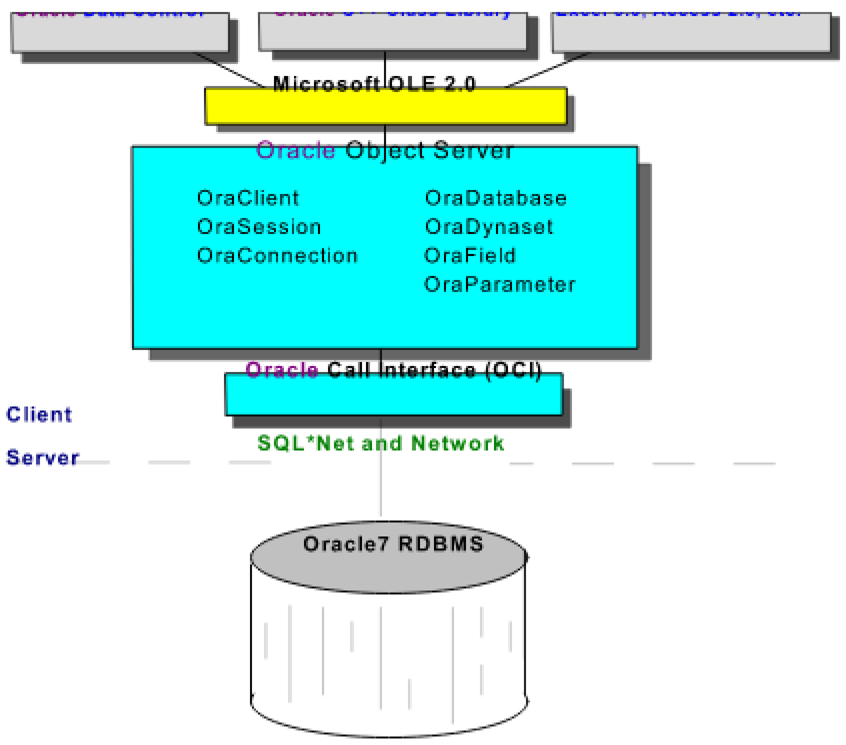
Oracle Objects for OLE, OLE, and the Oracle Database

Figure 1 shows the high-level relationship between the *Oracle Data Control,* the *Oracle Objects for OLE C++ Class Library*, the *Oracle Object Server,* OLE, and the Oracle7 database.

See Also

Oracle Data Control
Oracle Object Server
OraConnection Object
OraConnections Collection
OraClient Object
OraDatabase Object
OraDynaset Object
OraField Object
OraFields Collection
OraParameter Object
OraParameters Collection
OraSession Object
OraSessions Collection

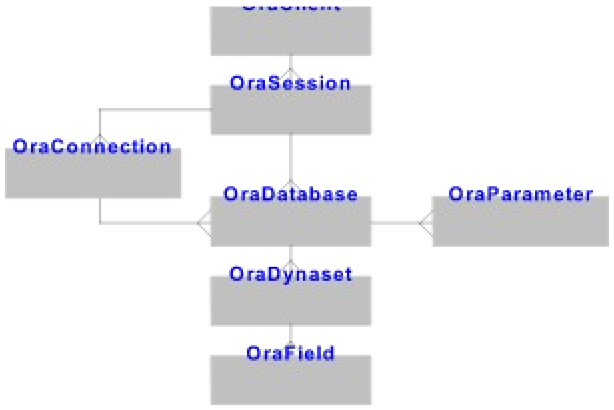
Figure 1 Relationships



Object Relationships

See Also

An operational hierarchy of the objects expresses has-a and "belongs-to" relationships. This hierarchy can be drawn as follows:



The "crow's feet" indicate the many ends of one-to-many or many-to-one relationships.

Each **OraClient** object can have many **OraSession** objects.
Each **OraSession** object can be associated with only one **OraClient** object.

Each **OraSession** object can have many **OraConnection** objects.
Each **OraConnection** object *can* be shared by many **OraDatabase** objects, although these OraDatabase objects must be within the same **OraSession** object.

Each **OraDatabase** object belongs to only one **OraSession** object.
Each **OraDynaset** object belongs to only one **OraDatabase** object.

Each **OraField** object belongs to only one **OraDynaset** object.
Each **OraParameter** object belongs to only one **OraDatabase** object.

You can create the **OraSession**, **OraDatabase**, **OraDynaset**, and **OraParameter** objects explicitly; the **OraClient**, **OraConnection**, and **OraFields** objects are created implicitly when necessary.

One **OraClient** object exists per workstation. This object is created when the first **OraSession** object is created. An **OraConnection** object may be created when an **OraDatabase** is created if that **OraDatabase** is not sharing a previously created connection. One **OraField** object is created per database column of your SQL SELECT statement when an **OraDynaset** is created. **OraParameter** objects are user created.

See Also

OraClient Object
OraConnection Object
OraDatabase Object
OraDynaset Object
OraField Object
OraParameter Object
OraSession Object

Getting Started

See Also

Oracle Objects for OLE is designed to provide access to the data of an Oracle database using the dynaset object.   The dynaset object is not the topmost object according to the Object Relationship diagram of Oracle Objects for OLE. This means that you must create or instantiate all of the objects a dynaset depends on (client, session, connection, and database)

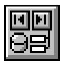
Using OLE Automation

When accessing the *Oracle Object Server* using the OLE Automation interface, you must create each object explicitly (except for the client object, which is always created automatically). The following code fragment demonstrates how to create first all of the objects required by a dynaset and then the dynaset itself.

```
...
'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object. Notice that this is the
'only object created via the CreateObject method. The
'argument to CreateObject is the name by which the
'OraSession object is known to the OLE system.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a
'connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb",
"scott/tiger", 0&)
'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp",
0&)

'You can now display or manipulate the data in the dynaset
'named 'OraDynaset'.
...
```

Using the Oracle Data Control

If you want to use the *Oracle Data Control* with Visual Basic 3.0, you should create a new project and then use the "Add File" option of the File menu to add the file ORADC.VBX to your project.   The *Oracle Data Control* will be added to your Visual Basic tool palette and will look like this:



The Oracle Data Control makes creating a dynaset easier, because it does not require you to create the underlying objects.   When the *Oracle Data Control* is refreshed, a client (if needed), session, database, and dynaset are created automatically.   The following code fragment demonstrates how to programmatically set the properties of the *Oracle Data Control* required to create a dynaset. Alternatively, you could set these properties by way of the properties window of Visual Basic.

```
...
 'Set the username and password.
 oradata1.Connect = "scott/tiger"

 'Set the databasename.
 oradata1.DatabaseName = "ExampleDb"

 'Set the recordsource.
 oradata1.RecordSource = "select * from emp"

 'Refresh the data control.
 oradata1.Refresh
...
```

You now have a valid client, session, database, and dynaset.   The client, session, database, and dynaset are referenced using oradata1.database.session.client, oradata1.database.session, oradata1.database, and oradata1.recordset, respectively.

Using Microsoft C++ or Borland C++

Please read the Oracle Objects for OLE C++ Class Library documentation for further details of using C++ with Oracle Objects for OLE.

See Also

Object Relationships
Oracle Data Control
OraClient Object
OraConnection Object
OraDatabase Object
OraDynaset Object
OraField Object
OraSession Object

Error Handling

See Also

OLE Automation Errors

The programmatic interface of the Oracle Object Server is OLE Automation. Therefore, errors that occur during execution of methods are frequently reported simply as an OLE Automation Error (ERR = 440, ERROR$ = OLE Automation Error). If such an error occurs, you should check the **LastServerErr** property of the **OraSession** and **OraDatabase** objects to determine whether an Oracle database error has occurred. If **LastServerErr** is non-zero, then an error has been raised by the Oracle Object Server. To find the Oracle Object Server Error, scan the string returned by the ERROR$ function for the string "OIP-NNNN" where NNNN is some error number included in the table below.

Constant	Value	Description
OERROR_ADVISEULINK	4096	Internal Error.
OERROR_POSITION	4098	An attempt was made to retrieve a field value from an empty dynaset.
OERROR_NOFIELDNAME	4099	An invalid field name was specified.
OERROR_TRANSIP	4101	A **BeginTrans** was specified while a transaction is already in progress
OERROR_TRANSNIPC	4104	A **CommitTrans** was specified without first executing **BeginTrans**.
OERROR_TRANSNIPR	4105	A **Rollback** was specified without first executing **BeginTrans**.
OERROR_NODSET	4106	Internal Error.
OERROR_INVROWNUM	4108	An attempt was made to reference an invalid row. This will happen when **EOF** or **BOF** is True or when the current row has been deleted and no record movement has occurred.
OERROR_TEMPFILE	4109	An error occurred while trying to create a temporary file for data caching.
OERROR_DUPSESSION	4110	An attempt was made to create a named session using **CreateSession** or **CreateNamedSession** that already exists.
OERROR_NOSESSION	4111	Internal Error.
OERROR_NOOBJECTN	4112	An attempt was made to reference a named object of a collection (other than the fields collection) that does not exist.
OERROR_DUPCONN	4113	Internal Error.
OERROR_NOCONN	4114	Internal Error.
OERROR_BFINDEX	4115	An invalid field index was specified. The range of indices is 0 to Count -1.
OERROR_CURNREADY	4116	Internal Error.
OERROR_NOUPDATES	4117	An attempt was made to change the data of a nonupdatable dynaset.
OERROR_NOTEDITING	4118	An attempt was made to change a fields value without first executing **Edit**.
OERROR_DATACHANGE	4119	An attempt was made to Edit data in the local cache, but the data on the Oracle server has been changed.
OERROR_NOBUFMEM	4120	Out of memory for data binding buffers.

OERROR_INVBKMRK	4121	An invalid bookmark was specified.
OERROR_BNDVNOEN	4122	Internal Error.
OERROR_DUPPARAM	4123	An attempt was made to create a named parameter using **Add**, but that name already exists.
OERROR_INVARGVAL	4124	An invalid offset or length parameters was passed to **GetChunk** or an internal error has occurred using **AppendChunk**.
OERROR_INVFLDTYPE	4125	An attempt was made to use **GetChunk** or **Append Chunk** on a field that was not of the type Long or Long Raw.
OERROR_TRANSFORUP	4127	A SELECT ... FOR UPDATE was specified without first executing **BeginTrans**
OERROR_NOTUPFORUP	4128	A SELECT ... FOR UPDATE was specified but the query is non-updatable.
OERROR_TRANSLOCK	4129	A **Commit** or **Rollback** was executed while a SELECT ... FOR UPDATE is in progress.
OERROR_CACHEPARM	4130	An invalid cache parameter was specified.
OERROR_FLDRQROWID	4131	An attempt was made to reference a field that requires a ROWID (Long or Long Raw), but the ROWID was not available.

These values can be found in the file **ORACONST.TXT**.

Oracle Errors

The most recent Oracle database error and error text is available from the **OraSession** or **OraDatabase** object properties **LastServerErr** and **LastServerErrText**.

The **LastServerErr** and **LastServerErrText** properties of the OraSession object return all errors related to connections, such as errors on **OpenDatabase**.

The **LastServerErr** and **LastServerErrText** properties of the OraDatabase object return all errors related to an Oracle cursor, such as errors on **CreateDynaset** and **ExecuteSQL**.

See Also

Add Method
AppendChunk Method
BeginTrans Method
BOF Property
CommitTrans Method
CreateNameSession Method.
CreateSession Method
Edit Method
EOF Property
ExecuteSQL Method
GetChunk Method
LastServerErr Property
LastServerErrPos Property
LastServerErrText Property
Locks and Editing
Long and Long Raw Columns
OpenDatabase Method
OraDatabase Object
OraSession Object
Rollback Method
SELECT ... FOR UPDATE

Locks and Editing

See Also

One of the defining features of client-server computing is that many clients may be accessing the server simultaneously.   This feature means that several clients may access the same table or record simultaneously.   In Oracle7 this issue is generally resolved using locks.   Locks allow one client to restrict other clients use of a table or record.   Locks are placed temporarily on database entities to prevent confusion and data corruption.

When you use Oracle Objects for OLE, locks are not placed on data until an **Edit** method is executed.   The **Edit** method attempts to obtain a lock (using "SELECT ... FOR UPDATE") on the current record of the dynaset.   This is done as late as possible to minimize the time that locks are placed on the records.   The **Edit** method can fail for several reasons:

- The SQL query violates the Oracle SQL updatability rules, for instance, by using calculated columns or table joins.
- The user does not have the privileges needed to obtain a lock.
- The record has been locked already by another user.   The **OpenDatabase** method has an option so that you can decide whether to wait on locks.

See Also

Edit Method
OpenDatabase Method
Using "SELECT ... FOR UPDATE"

Transactions

See Also

A transaction is a logical unit of work that comprises one or more SQL statements executed by a single user. A typical example is transferring money from one bank account to another. Two operations take place:

1. Money is taken out of one account.
2. Money is put into the other account.

These operations need to be performed together. If one were to be done and the other not done (for example, if the network connection went down), the banks books would not balance correctly.

Normally, when you execute an **Update** method on a dynaset, the changes are committed to the database immediately. Each operation is treated as a distinct transaction. Using the **BeginTrans**, **CommitTrans**, and **Rollback** transactional control methods of the **OraSession** object allows operations to be grouped into larger transactions. **BeginTrans** tells the session that you are starting a group of operations. **CommitTrans** makes the entire group of operations permanent. **Rollback** cancels the entire group. **CommitTrans** and **Rollback** end the transaction and the program returns to normal operation: one transaction per operation. Experienced Oracle users should note the following differences between the operation of Oracle Objects for OLE and many Oracle tools:

- Oracle tools such as SQL*Plus execute as if the **BeginTrans** method was called when the tool was started. This means that updates are not committed immediately, but are held until a commit or rollback is executed.
- SQL*Plus always starts a new transaction every time a commit or rollback is executed.
- The autocommit setting in SQL*Plus results in behavior similar to the default of the Oracle Objects for OLE.

If you are connected to more than one database and use the transaction methods, you should understand that Oracle Objects for OLE commits each database separately. This is *not* the same as the two-phase commit that Oracle7 provides. If your application needs to guarantee data integrity across databases, you should connect to a single database and then access additional databases by way of the Oracle7 database link feature. This method gives you the benefit of Oracle7s two-phase commit. Consult your Oracle7 documentation for more information about two-phase commit, database links, and distributed transactions.

Transactions apply only to the Data Manipulation Language (DML) portion of the SQL language (such as INSERT, UPDATE, and DELETE). Transactions do not apply to the Data Control Language (DCL) or Data Definition Language (DDL) portions (such as CREATE, DROP, ALTER, etc.) of the SQL language. DCL and DDL commands always force a commit, which in turn commits everything done before them.

See Also

BeginTrans Method
CommitTrans Method
OraConnection Object
OraSession Object
ResetTrans Method
Rollback Method

Long and Long Raw Columns

See Also

Putting

Long and long raw columns of an Oracle database can contain up to 2 gigabytes of data. Data of less than 64K bytes can be put into the database using simple assignment, but data exceeding 64K bytes must be put into the database using **AppendChunk**. The Oracle database libraries do not allow putting of data in multiple pieces, so the data is held internally and put in one piece upon an **Update**.

Fetching

Long and long raw columns of an Oracle database can contain up to 2 gigabytes of data. This makes it impractical to retrieve all data from a long or long raw column automatically when it is selected. Instead, the first 64K bytes is retrieved and the Oracle ROWID is cached locally so that the row containing the long or long raw column can be located and then retrieved when using the **GetChunk** method. If the Oracle ROWIDs cannot be obtained on a query with a long or long raw column, then the long or long raw column will not be accessible and an error will be generated if field access is attempted.

Editing

Before an **Edit** can be started, a column's locally cached value is compared with its current database value. If the values match, the edit proceeds; otherwise an error is generated. Since long and long raw columns may contain up to 2 gigabytes of data, *no* comparison is done before an **Edit** is started.

See Also

- **AppendChunk** Method
- **FieldSize** Method
- **Edit** Method
- **GetChunk** Method
- **OraDynaset** Object
- **Update** Method

"SELECT ... FOR UPDATE"

See Also Example

Normally, when a dynaset is created, rows are not locked in the database until **Edit** is invoked. If this is not desirable, you might include the FOR UPDATE construct in the SQL SELECT statement. Unfortunately, the FOR UPDATE construct undermines normal dynaset operations, so Oracle does *not* recommend its use.

Dynasets created with FOR UPDATE are handled correctly in most cases by scanning the SQL statement for the FOR UPDATE construct (This is necessary because the Oracle database functions do not distinguish between SELECT and SELECT FOR UPDATE SQL statements.) It is possible that some exotic FOR UPDATE SQL statement will be treated as "not FOR UPDATE"meaning that rows are not locked during the lifetime of the dynaset. If the FOR UPDATE is not recognized, rows are locked only during an **Edit**/**Update** sequence. However, during the **Edit**/**Update** sequence, the row is verified as unchanged before the **Edit** is permitted.

The use of FOR UPDATE on dynasets requires that a session level transaction be in progress at the time the dynaset is created. Further, before the session can be committed or rolled back, all objects which reference the dynaset must be set to "Nothing" or an error is returned. In the case of a data control, change the record source and **Refresh** the data control or **Recordset**.

Note that if an error results and the application terminates, uncommitted data is rolled back, including pending FOR UPDATE dynasets.

See Also

Edit Method
Locks and Editing
Refresh Method
Recordset Property
Update Method

"SELECT ... FOR UPDATE" Example

Close Copy Print

This example demonstrates the use of SELECT ... FOR UPDATE to lock all the rows of a dynaset while it is being updated.   Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object
 Dim fld As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Start Transaction processing before creating the dynaset
 'with FOR UPDATE, or an error will occur.
 OraSession.DbBeginTrans

 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp FOR UPDATE",
0&)

 ' Create a field object for faster access.
 ' This will cause a reference to the Dynaset to be held.
 Set fld = OraDynaset.Fields("sal")

 'Traverse until EOF is reached, setting
 'each employee's salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   fld.value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'When using FOR UPDATE, you must close (reduce the
 ' reference count to zero) the dynaset by setting it to
 ' Nothing, or an error will occur.
 Set OraDynaset = Nothing

 'You must also set fld to Nothing since it contains a
 ' reference to the dynaset.
 Set fld = Nothing

 'End Transaction processing
 OraSession.DbCommitTrans

End Sub
```

Tuning and Customization

See Also

A number of working parameters of Oracle Objects for OLE can be customized. Access to these parameters is provided through the Oracle initialization file, by default named ORAOLE.INI. Each entry currently available in that file is described below. The location of the ORAOLE.INI file is specified by the ORAOLE environment variable. Note that this variable should specify a full pathname to the Oracle initialization file, which is not necessarily named ORAOLE.INI. If this environment variable is not set, or does not specify a valid file entry, then Oracle Objects for OLE looks for a file named ORAOLE.INI in the Windows directory. If this file does not exist, all of the default values listed will apply.

You can customize the following sections of the ORAOLE.INI file:

[Cache Parameters]

A cache consisting of temporary data files is created to manage amounts of data too large to be maintained exclusively in memory. This cache is needed primarily for dynaset objects, where, for example, a single LONG RAW column can contain more data than exists in physical (and virtual) memory.

The default values have been chosen for simple test cases, running on a machine with limited Windows resources. Tuning with respect to your machine and applications is recommended.

Note that the values specified below are for a single cache, and that a separate cache is allocated for each object that requires one. For example, if your application contains three dynaset objects, three independent data caches are constructed, each using resources as described below.

SliceSize = 256 (default)
This entry specifies the minimum number of bytes used to store a piece of data in the cache. Items smaller than this value are allocated the full *SliceSize* bytes for storage; items larger than this value are allocated an integral multiple of this space value. An example of an item to be stored is a field value of a dynaset.

PerBlock = 16 (default)
This entry specifies the number of *Slices* (described in the preceding entry) that are stored in a single block. A block is the minimum unit of memory or disk allocation used within the cache. Blocks are read from and written to the disk cache temporary file in their entirety. Assuming a *SliceSize* of 256 and a *PerBlock* value of 16, then the block size is 256 * 16 = 4096 bytes.

CacheBlocks = 20 (default)
This entry specifies the maximum number of blocks held in memory at any one time. As data is added to the cache, the number of used blocks grows until the value of *CacheBlocks* is reached. Previous blocks are swapped from memory to the cache temporary disk file to make room for more blocks. The blocks are swapped based upon recent usage. The total amount of memory used by the cache is calculated as the product of (*SliceSize* * *PerBlock* * *CacheBlocks*).

Recommended Values: You may need to experiment to find optimal cache parameter values for your applications and machine environment.   Here are some guidelines to keep in mind when selecting different values:

- The larger the (*SliceSize* * *PerBlock*) value, the more disk I/O is required for swapping individual blocks.
- The smaller the (*SliceSize* * *PerBlock)* value, the more likely it is that blocks will need to be swapped to or from disk.
- The larger the *CacheBlocks* value, the more memory is required, but the less likely it is that swapping will be required.

A reasonable experiment for determining optimal performance might proceed as follows:

- Keep the *SliceSize* >= 128 and vary *PerBlock* to give a range of block sizes from 1K through 8K.
- Vary the *CacheBlocks* value based upon available memory.   Set it high enough to avoid disk I/O, but not so high that Windows begins swapping memory to disk.
- Gradually decrease the *CacheBlocks* value until performance degrades or you are satisfied with the memory usage.   If performance drops off, increase the *CacheBlocks* value once again as needed to restore performance.

[Fetch Parameters]

FetchLimit = 20 (default)

This entry specifies the number of elements of the array into which data is fetched from Oracle.   If you change this value, all fetched values are <u>immediately</u> placed into the cache, and all data is retrieved from the cache. Therefore, you should create cache parameters such that <u>all</u> of the data in the fetch arrays can fit into cache memory. Otherwise, inefficiencies may result.

Increasing the *FetchLimit* value reduces the number of fetches (calls to the database) calls and possibly the amount of network traffic.   However, with each fetch, more rows must be processed before user operations can be performed.   Increasing the *FetchLimit* increases memory requirements as well.

FetchSize = 4096 (default)

This entry specifies the size, in bytes, of the buffer (string) used for retrieved data.   This buffer is used whenever a long or long raw column is initially retrieved.

[General]

TempFileDirectory = [*Path*]

This entry provides one method for specifying disk drive and directory location for the temporary cache files.   The files are created in the first legal directory path given by:

1. The drive and directory specified by the TMP environment variable (this method takes precedence over all others);
2. The drive and directory specified by this entry (*TempFileDirectory*) in the [general] section of the ORAOLE.INI file;
3. The drive and directory specified by the TEMP environment variable; or
4. The current working drive and directory.

HelpFile = [*Path and File Name*]

This entry specifies the full path (drive/path/filename) of the Oracle Objects for OLE help file as needed by the Oracle Data Control.   If this entry cannot be located, the file ORACLEO.HLP is assumed to be in the directory where ORADC.VBX is located (normally \WINDOWS\SYSTEM).

See Also

CreateDynaset Method
Long and Long Raw Columns
OraDynaset Object

Coding Techniques

See Also

Avoiding Excessive Object References

Oracle Objects for OLE is based on OLE Automation. While OLE Automation provides an excellent method of extending applications, it is not without cost. Poor coding techniques can cause extra object references and result in poor performance.

When using OLE Automation, it is preferable to reduce the number of object references. If you find you are frequently referencing a particular object (a column for example), it is much faster to declare a second object and set it to the value of the much-referenced object. This is true for singular objects as well as objects that are part of a collection.

Consider the following code fragment:

```
...
'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

 'Create the OraSession Object
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb",
"scott/tiger", 0&)

 'Create the OraDynaset Object
 Set OraDynaset = OraSession.Database.DbCreateDynaset("select *
from emp", 0&)
```

The dynaset creation above is inefficient, because the **Database** property of the **OraSession** object is used instead of using the **OraDatabase** object that was already created. That code results in one extra object reference. The correct code is:

```
 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp",
0&)
```

This example is only a small improvement and will not likely result in much time saved. Consider the following code fragment:

```
...
 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp",
0&)

'Traverse until EOF is reached, setting each employee's
 'salary and commission to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.Fields("comm").value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
...
```
The field references are inefficient, because the **Fields** property of the **OraDynaset** object is referenced every time a field is needed. The correct code is:
```
...
Dim flds() As Object
Dim i,fldcount As Integer

 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 
0&)

'Get the field count, and output the names
 fldcount = OraDynaset.Fields.Count

 ReDim flds(0 To fldcount - 1)
 For i = 0 To fldcount
  Set flds(i) = OraDynaset.Fields(i)
 Next I

 'Traverse until EOF is reached, setting each employee's
 'salary and commission to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   flds(5).value = 0    'SAL is column 5 (zero based).
   flds(6).value = 0    'COMM is column 6 (zero based).
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
...
```
This examples shows how to reference fields through a field object and not through the fields collection of the dynaset. Testing proves that this small amount of extra code greatly improves performance.

Any method or object that is referenced through more than one object is potentially inefficient, but the extra coding is not always worth the time saved (as in the first example above). The best place to start is with field references, because they are most likely to occur multiple times.

See Also

Database Property
Fields Property
OraDatabase Object
OraField Object
OraFields Collection
OraSession Object

Sample Code Conventions

See Also

The following conventions apply throughout the sample code included here and the sample applications shipped with Oracle Objects for OLE:

- The user scott with password tiger (scott/tiger) is used for the **Connect** property.
- The SQL*Net alias ExampleDb is used for the **DatabaseName** property.
- The data tables referenced are the standard Oracle demonstration tables. You can create them with the script **DEMOBLD7.SQL**. (You can drop these tables and views with the script **DEMODRP7.SQL**.)
- You can create the stored procedures referenced with the script **ORAEXAMP.SQL**.
- The file **ORACONST.TXT** contains constant values such as **True** and **False** (defined as the corresponding values in Visual Basic), and other constants used for options flags and property values.
- If the Oracle Data Control is used, its name is "oradata1".

See Also

Connect Property
DatabaseName Property

Method And Property Name Conflicts

See Also

The Oracle Objects for OLE programmatic interface is provided via OLE Automation. In some cases, the method or property names used in Oracle Objects for OLE may conflict with reserved words or built-in function names of some applications. If this is the case, you can prefix ANY Oracle Objects for OLE method or property name with "Db". All known application conflicts are listed below.

Visual Basic 3.0

The method names listed below are reserved words in Visual Basic 3.0. When Visual Basic 3.0 performs syntax checking, an error will be generated if these methods are not being referenced using a standard data object (of type "database" or "dynaset"). Therefore, in Visual Basic 3.0, these methods can only be used with the "Db" prefix.

Method	Applicable Object	Example
AddNew	OraDynaset	oradyanset.DbAddNew
AppendChunk	OraField	orafield.DbAppendChunk
BeginTrans	OraSession	orasession.DbBeginTrans
Clone	OraDynaset	oradynaset.DbClone
CommitTrans	OraConnection	oraconnection.DbCommitTrans
	OraSession	orasession.DbCommitTrans
CreateDynaset	OraDynaset	oradynaset.DbCreateDynaset
Delete	OraDynaset	oradynaset.DbDelete
Edit	OraDynaset	oradynaset.DbEdit
ExecuteSQL	OraDatabase	oradynaset.DbExecuteSQL
FieldSize	OraDynaset	oradynaset.DbFieldSize
GetChunk	OraField	orafield.DbGetChunk
MoveFirst	OraDynaset	oradynaset.DbMoveFirst
MoveLast	OraDynaset	oradynaset.DbMoveLast
MoveNext	OraDynaset	oradynaset.DbMoveNext
MovePrevious	OraDynaset	oradynaset.DbMovePrevious
Refresh	OraDynaset	oradynaset.DbRefresh
Rollback	OraConnection	oraconnection.DbRollback
	OraSession	orasession.DbRollback
Update	OraDynaset	oradynaset.DbUpdate
UpdateControls	Data Control Recordset	oradata1.recordset.DbUpdateControls
UpdateRecord	Data Control Recordset	oradata1.recordset.DbUpdateRecord

Access 2.0

The method names listed below are built-in function names in Access 2.0. When Access 2.0 compiles a module, an error will be generated if these methods are not being referenced using a standard data object (of type "database" or "dynaset"). Therefore, in Access 2.0, these methods can only be used with the "Db" prefix.

Method	Applicable Object	Example
AddNew	OraDynaset	oradyanset.DbAddNew
AppendChunk	OraField	orafield.DbAppendChunk

BeginTrans	OraSession	orasession.DbBeginTrans
Clone	OraDynaset	oradynaset.DbClone
CommitTrans	OraConnection	oraconnection.DbCommitTrans
	OraSession	orasession.DbCommitTrans
CreateDynaset	OraDynaset	oradynaset.DbCreateDynaset
Delete	OraDynaset	oradynaset.DbDelete
Edit	OraDynaset	oradynaset.DbEdit
ExecuteSQL	OraDatabase	oradatabase.DbExecuteSQL
FieldSize	OraDynaset	oradynaset.DbFieldSize
GetChunk	OraField	orafield.DbGetChunk
MoveFirst	OraDynaset	oradynaset.DbMoveFirst
MoveLast	OraDynaset	oradynaset.DbMoveLast
MoveNext	OraDynaset	oradynaset.DbMoveNext
MovePrevious	OraDynaset	oradynaset.DbMovePrevious
OpenDatabase	OraSession	orasession.DbOpenDatabase
Refresh	OraDynaset	oradynaset.DbRefresh
Rollback	OraConnection	oraconnection.DbRollback
	OraSession	orasession.DbRollback
Update	OraDynaset	oradynaset.DbUpdate

See Also

AddNew Method
AppendChunk Method
BeginTrans Method
Clone Method
CommitTrans Method
CreateDynaset Method
Delete Method
Edit Method
ExecuteSQL Method
FieldSize Method
GetChunk Method
MoveFirst Method
MoveLast Method
MoveNext Method
MovePrevious Method
OpenDatabase Method
Refresh Method
Rollback Method
Update Method
UpdateControls Method
UpdateRecord Method

Redistributable Files

Oracle Objects for OLE

The following redistributable files are part of Oracle Objects for OLE and must be distributed with your application developed using Oracle Objects for OLE:

ORAIPSRV.DLL
ORAIPSRV.REG
ORAIPSRV.TLB

These files should be installed in the \WINDOWS\SYSTEM directory. In addition to including these three files with your application, you must register in the Windows registration database the information found in ORAIPSRV.REG. Use REGEDIT (included with Windows) to do this.

Finally, you must also distribute the file from the following list corresponding to the development software you used to build your application:

ORACLB.DLL (for Borland C++ 4.0)
ORACLB45.DLL (for Borland C++ 4.5)
ORACLM.DLL (for Microsoft C++)
ORADC.VBX (for Visual Basic 3.0)

Microsoft OLE 2.0

The following files are part of Microsoft OLE 2.0 and are required by Oracle Objects for OLE. Please refer to the OLE 2 Programmer's Reference or the Visual Basic documentation for details on installing and distributing these files.

COMPOBJ.DLL
OLE2.DLL
OLE2.REG
OLE2CONV.DLL
OLE2DISP.DLL
OLE2NLS.DLL
OLE2PROX.DLL
STDOLE.TLB
STORAGE.DLL
TYPELIB.DLL

Visual Basic 3.0

The following files are part of Visual Basic and are required by Oracle Objects for OLE. If you are shipping a Visual Basic application that uses Oracle Objects for OLE, you should include at least the following files (in addition to other Visual Basic runtime files that you may need). Please refer to the Visual Basic documentation for details on installing and distributing these files.

VBOA300.DLL
VBRUN300.DLL

Troubleshooting

See Also

OLE Initialization or OLE Automation Errors

The most frequent cause of OLE Initialization and Automation errors is missing or incorrectly installed software. Please ensure correct installation of the software specified below. Then make sure that you have specified method and property names correctly and that you have declared all "Oracle objects" as type "object".

Possible Cause	**Solution**
Your system does not contain the Microsoft OLE 2.0 runtime files or these files are out of date. Note that Visual Basic 3.0 does not include the OLE file TYPELIB.DLL.	Reinstall Oracle Objects for OLE and select the "Microsoft OLE 2.0 Libraries" option.
OLE 2.0 information was not registered in the Windows registration database.	Run REGEDIT.EXE and merge the information from the file OLE2.REG (normally located in \WINDOWS\SYSTEM).
The Oracle Objects for OLE object information was not registered in the Windows registration database.	Reinstall Oracle Objects for OLE and select the "Oracle Objects Server" option.
The Oracle Objects for OLE object information was not registered in the Windows registration database.	Run REGEDIT.EXE and merge the information from the file ORAIPSRV.REG (normally located in \WINDOWS\SYSTEM).
Your system does not contain the Oracle Required Support Files (ORA71WIN.DLL, CORE3WIN.DLL, NLS23WIN.DLL, etc.) or these files are not on the PATH.	Reinstall Oracle Objects and select the "Oracle Required Support Files" option or add to your PATH environment variable the directory where these files are located.
Your system does not contain Oracle SQL*Net or its files are not on the PATH.	Install Oracle SQL*Net or add to your PATH environment variable the directory where these files are located.
You have misspelled a method or property name.	Check the documentation to determine the correct spelling.
You have referenced a method or property from the wrong object.	Check the documentation to determine the correct object.

Oracle SQL*Net Errors

The most frequent cause of Oracle SQL*Net errors is incorrectly specified connection information. The connection information for Oracle Objects for OLE is specified differently than when using ODBC. Please verify that you have specified connection information correctly, and then make sure your SQL*Net connection is working properly before using Oracle Objects for OLE. The appropriate Oracle SQL*Net documentation contains information about testing your connection and about any Oracle SQL*Net error that you may receive.

Possible Cause	**Solution**
Incorrect **Connect** property or argument to the **OpenDatabase** method.	See the topics on the **Connect** property or the **OpenDatabase** method for examples.
Incorrect **DatabaseName** property or argument to the **OpenDatabase** method.	See the topics on the **DatabaseName** property or the **OpenDatabase** method for examples.
Your system does not contain Oracle SQL*Net.	Install Oracle SQL*Net.

General Protection Faults

The most frequent cause of GPFs is installing Oracle Objects for OLE while other applications are running that require the Oracle Object Server, Oracle Required Support Files or OLE 2.0.   To avoid this, install Oracle Objects for OLE immediately after starting Windows and before running any other application.

Possible Cause	**Solution**
Duplicate Oracle Objects for OLE files exist in the \WINDOWS or \WINDOWS\SYSTEM directories or along the PATH.	Remove any duplicate files. The files ORAIPSRV.DLL, ORAIPSRV.TLB, and ORAIPSRV.REG should only be located in \WINDOWS\SYSTEM.
Duplicate Oracle Required Support Files DLLs exist in the \WINDOWS or \WINDOWS\SYSTEM directories or along the PATH.	Remove any duplicate files. Typically, the Oracle Required Support Files DLLs (ORA7*.DLL, CORE*.DLL, and NLS*.DLL) are located in \ORAWIN\BIN.
Duplicate OLE 2.0 DLLs exist in the \WINDOWS or \WINDOWS\SYSTEM directories or along the PATH.	Remove any duplicate files. The OLE 2.0 DLLs (listed in the Redistributable Files topic) should only be located in \WINDOWS\SYSTEM.

See Also

Connect Property
DatabaseName Property
OpenDatabase Method
Redistributable Files

Data Aware Controls

See Also

The Oracle Data Control is a fully functional Visual Basic 3.0 level 3 Custom Control and should be compatible with any data aware control that binds using the standard VBM_DATA messages found in VBAPI.H (included with the VB Control Development Kit).

In order to work properly with the Oracle Data Control, data aware controls should not rely upon the Oracle Data Control being a built-in data control or should not assume the window class name of the Oracle Data Control or that the properties can be cloned and a VB data control used in place of the Oracle Data Control.

Binding Messages

In particular, the Oracle Data Control will respond to VBM_DATA_GET, VBM_DATA_METHOD, VBM_DATA_SET, VBM_DATA_INITIATE and VBM_DATA_TERMINATE messages and will generate VBM_DATA_AVAILABLE and VBM_DATA_REQUEST messages.

If a bound control sends the VBM_DATA_GET message with sAction = DATA_FIELDSIZE the Oracle Data Control will only return a size of up to 64KB. If the field is larger than 64KB, the size returned will be -1 (negative 1). Bound controls should then send VBM_DATA_GET with sAction = DATA_FIELDCHUNK to retrieve chunks of the column. When a returned chunk size is smaller than the requested size and an error hasn't occurred, the field has been fully transferred. This behavior is similar to the implementation of the **GetChunk/AppendChunk** methods.

The Oracle Data Control doesn't respond to the undocumented VBM_DATA_FIND messages.

Picture Support

The Oracle Data Control responds to the VBM_DATA_GET message with sAction = DATA_FIELDVALUE and usDataType = DT_PICTURE to support picture controls that use this interface (most notably the VB picture and image controls). Note that not all picture controls work using this message. Some controls will fetch the data as a binary stream and manipulate it internally.

Field Types

It is important to note that certain bound controls expect a particular field type (and may not work with any other) when binding to a database column. The Oracle Data Control maps column types to fields types as is listed under the **Type** property.

Notes on Particular Data Aware Controls

The following data aware controls have been tested with the Oracle Data Control and the following comments are available. Testing may not have been comprehensive and even though a control is listed here, Oracle does not guarantee that it will work with the Oracle Data Control. Other controls not listed here should work with the Oracle Data Control as long as they follow the guidelines listed above.

TrueGrid by Apex Software Corporation

Version 2.0 and 2.1 as well as the layout editor work properly with the Oracle Data Control.

QuickPack Professional from Crescent Software

The Combo (CSCOMBO.VBX) and List box (CSVLIST.VBX) do not work with the Oracle Data Control because they assume the window class name of the data control they are bound to.   Also note that when comparing the DataSource property of the combo or list box to the name of the data control, a case sensitive comparison is done.

Search for the string ThunderData in cscombo.c (for the combo box) and ctl.c (for the list box) to see where the window class name is assumed (The window class name of the Oracle Data Control is OraData).   Search for _fstrcmp in those same files to see where a case sensitive name comparison is done.

If the combo box is rebuilt with the above problems fixed it will not work properly unless the Oracle Data Control named in the DataSourceList property of the combo box is refreshed before the Oracle Data Control named in the DataSource property.   The List box, even after fixing the above problems, still does not work.

It appears that the code for both the combo and list box do not differentiate between the DataSource and DataSourceList data controls and try to access them in the incorrect order before they are refreshed.

The Picture control (CSPICTUR.VBX) does not work with images greater than 64KB for the reasons described in the Binding Messages section above.

Aware, Grid and Spread by Farpoint Technologies

The Aware Memo control (AWAREMM.VBX), according to it's documentation, must be bound to a memo field.   This means that it must be bound to an Oracle column of type LONG since that is the only type that maps to a memo field.

Neither the Grid nor the Spread interface designers work with the Oracle Data Control.   Please contact Farpoint Technologies for possible upgrade information.

FX Tools by ImageFX

No comments available.

ImageKnife by Media Architects

Version 1.3 of this control does not work with images greater than 64KB for the reasons described in the Binding Messages section above.   Please contact Media Architects for possible upgrade information.

VBTools4 by Microhelp

Many of the controls provided attempt to update data even if no changes have been made and therefore do not work when the ReadOnly property of the Oracle Data Control is TRUE.   This behavior is not specific to the Oracle Data Control.

The Combo (MHGCMB.VBX) and List box (MHGLBX.VBX) do not work with the Oracle Data Control.

The Out of Bounds Control (MHGODB.VBX) does not work with the Oracle Data Control.

Data Widgets by Sheridan Software Systems

You must upgrade to Version 1.0d of the Data Widgets to work properly with the Oracle Data Control.

VBXTASY Volume 1 by Spinoza Limited

None of the Check controls work properly with the Oracle Data Control because they are requesting a field type of DATA_VT_VAR_BYTE which does not correspond to any column type to field type mapping made by Oracle Objects (as listed in the **Type** property).

Formula One By Visual Tools

Columns of type DATE do not display correctly.

Image Stream By Visual Tools

This control does not work with the Oracle Data Control.

See Also

AppendChunk Method
GetChunk Method
Type Property

Oracle Objects for OLE and Visual Basic

See Also

Data Access Objects/Methods/Properties

This topic compares the objects, methods and properties of Oracle Objects for OLE to the Data Access Objects (DAO) found in Visual Basic. This section should not be considered a replacement for the full reference, but rather a place to start when trying to determine if a feature of DAO is available in Oracle Objects for OLE.

Objects	**OO**	**VB**	**Comments**
Client	YES	NO	OO: One client object exists per workstation.
Connection	YES	NO	OO: Represents a single connection to the database
Database	YES	YES	
Dynaset	YES	YES	
Fields	YES	YES	
Index	N/A	YES	OO: Indexes are not required on tables.
Parameter	YES	NO	OO: Provides SQL and PL/SQL bind variable support.
QueryDef	NO	YES	OO: Queries are not saved in the database. Views should be used as an alternative.
Session	YES	NO	OO: Used to manage collections of databases, connections, and dynasets,
Snapshot	NO	YES 3	OO: Set the READONLY flag when creating a dynaset.
Table	NO	YES 3	OO: Tables are part of the data dictionary.
TableDef	NO	YES	OO: Table definitions are part of the data dictionary.

NOTE: "Oracle objects" are declared as data type (OLE) "object", not as native data types.

Client Methods	**OO**	**VB**	**Comments**
CreateSession	YES	N/A	OO: Creates a named session that can be located by name.

Client Properties	**OO**	**VB**	**Comments**
Name	YES	N/A	OO: Returns the name of the Client object.
Sessions	YES	N/A	OO: Returns the associated Sessions collections.

Session Methods	**OO**	**VB**	**Comments**
BeginTrans1	YES	N/A	OO: Begins a Session-wide transaction.
CreateNamedSession	YES	N/A	OO: Same as Client.CreateSession.
ConnectSession	YES	N/A	OO: Same as finding a session in the Client.Sessions collection.
CommitTrans1	YES	N/A	
OpenDatabase	YES	N/A	VB: OpenDatabase is a standalone method.
LastServerErrReset	YES	N/A	OO: Clears the error in LastServerErr to zero and sets LastServerErrText to NULL.
Rollback1	YES	N/A	OO: Rolls back a Session-wide transaction.
ResetTrans	YES	N/A	OO: Unconditionally rolls back a Session-wide transaction.

Session Properties	**OO**	**VB**	**Comments**

Name	YES	N/A	OO: Returns the Session name.
Client	YES	N/A	OO: Returns the associated Client object.
Connections	YES	N/A	OO: Returns the associated Connections collection.
LastServerErr	YES	N/A	OO: Returns the connection related error.
LastServerErrText	YES	N/A	OO: Returns the text of the last connection related error.
OIPVersionNumber	YES	N/A	OO: Returns the current internal version number.

Connection Properties	**OO**	**VB**	**Comments**
Connect	YES	N/A	OO: Returns the Connect property minus the password.
DatabaseName	YES	N/A	OO: Returns the DatabaseName property.
Session	YES	N/A	OO: Returns the associated Session object.

Connection Methods	**OO**	**VB**	**Comments**
CommitTrans	YES 1	N/A	OO: Commits transactions at the connection level.
Rollback	YES 1	N/A	OO: Rolls back transactions at the connection level.

Database Properties	**OO**	**VB**	**Comments**
CollatingOrder	NO	YES	VB: Determines the method for comparing text values. OO: Use 'ALTER SESSION ... NLS_SORT ...
Connect	YES	N/A	OO: Same as Connection.Connection.
Connection	YES	NO	OO: Returns the Connection object.
DatabaseName	YES	NO	OO: Same as Connection.DatabaseName.
Exclusive	N/A	YES	OO: Clients generally have no control over database access.
LastServerErr	YES	NO	OO: Returns the number of the last cursor related error.
LastServerErrPos	YES	NO	OO: Returns the position in the SQL statement that a parse error occurred.
LastServerErrText	YES	NO	OO: Returns the text of the last cursor related error.
Name	NO	YES	
Options	YES	NO	OO: Returns the options flag originally passed to the Session.OpenDatabase method.
Parameters	YES	N/A	OO: Returns the associated parameters collection.
QueryDefs	N/A	YES	
QueryTimeout	NO	YES	VB: Determines the number of seconds Visual Basic waits before a timeout error occurs when executing a query on an ODBC database. OO: See the Lock Wait option of the OpenDatabase method.
Session	YES	N/A	OO: Returns the associated Session object.
Transactions	NO	YES	OO: This flag is available through the Dynaset Object.
Updatable	NO	YES	OO: This flag is available through the Dynaset Object.

Database Methods	**OO**	**VB**	**Comments**
BeginTrans	NO	YES 3	OO: This method is available via the Session Object.
Close	N/A	YES	OO: Once the database has gone out of scope and

			there are no references to it, it closes automatically.
CommitTrans	NO	YES 3	OO: This method is available via the Session Object.
CreateDynaset	YES 1	YES 3	The options flags for Oracle and VB CreateDynaset methods are significantly different. Please consult the respective documentation for details.
CreateQueryDef	N/A	YES	
CreateSnapshot	N/A	YES	OO: Set the READONLY property when creating the dynaset.
DeleteQueryDef	N/A	YES	
Execute	N/A	YES	OO: Use ExecuteSQL.
ExecuteSQL	YES 1	YES	
LastServerErrReset	YES	N/A	OO: Clears the error in LastServerErr to zero and sets LastServerErrText to NULL.
ListFields	NO	YES	
ListTables	NO	YES	
OpenQueryDef	N/A	YES	
OpenTable	N/A	YES	
Rollback	NO	YES 2	OO: This method is available through the Session Object.

<u>Parameter Properties</u>	**OO**	**VB**	**Comments**
MinimumSize	YES	N/A	OO: Minimum size of a parameter (for binding).
Name	YES	N/A	OO: Name of the parameter.
ServerType	YES	N/A	OO: Specifies the Oracle Data Type.
Status	YES	N/A	OO: Bit flag to indicate state (Input/Output/AutoBound/Enabled).
Type	YES	N/A	OO: Indicates the variant data type that is actually bound to the SQL statement.
Value	YES	N/A	OO: Parameter value.

<u>Parameter Methods</u>	**OO**	**VB**	**Comments**
Add	YES	N/A	OO: Add to the Parameter collection.
AutoBindEnable	YES	N/A	OO: Bind on Dynaset Create/Refresh.
AutoBindDisable	YES	N/A	OO: Do Not Bind on Dynaset Create/Refresh.
Remove	YES	N/A	OO: Remove from the Parameter collection.

<u>Dynaset Methods</u>	**OO**	**VB**	**Comments**
AddNew	YES 1	YES	
Clone	YES 1	YES 3	
Close	N/A	YES	OO: Once the dynaset has gone out of scope and there are no references to it, it will be closed automatically.
Delete	YES 1	YES	
Edit	YES 1	YES	
FindFirst	NO	YES	
FindLast	NO	YES	
FindNext	NO	YES	
FindPrevious	NO	YES	

ListFields	NO	YES 1	VB: Suggests that this method should not be used.
ListIndexes	NO	YES 3	VB: Suggests that this method should not be used.
MoveFirst	YES 1	YES	
MoveLast	YES 1	YES	
MoveNext	YES 1	YES	
MovePrevious	YES 1	YES	
Refresh	YES 1	NO	OO: Cancels all edit operations, executes the current contents of the SQL statement buffer, and moves to the first row of the resulting dynaset. Used to re-execute a SQL statement where only a parameter value has changed.
Update	YES 1	YES	
UpdateControls	YES 1	NO	VB: Implemented as a Data Control method.
UpdateRecord	YES 1	NO	VB: Implemented as a Data Control method.

<u>Dynaset Properties</u>	**OO**	**VB**	**Comments**
BOF	YES	YES	
Bookmark	YES	YES	
Bookmarkable	YES	YES	OO: All dynasets are bookmarkable except if created with the NoCache option.
Connection	YES	N/A	OO: Returns the associated Connection object.
Database	YES	NO	OO: Returns the associated Database object.
DateCreated	N/A	YES	OO: Check the Data Dictionary.
EOF	YES	YES	
EditMode	YES	YES	
Fields	YES	YES	
Filter	NO	YES 3	OO: Use a WHERE clause.
Index	NO	YES	
LastModified	YES	YES	
LastUpdated	N/A	YES	OO: Check the Data Dictionary.
LockEdits	N/A	YES	OO: Rows are locked when updated or if the SQL statement contains a 'FOR UPDATE' clause.
Name	NO	YES	VB: The name of the Dynaset is either the Name property of a Table or QueryDef used to create it, or the SQL statement used to create it.
NoMatch	N/A	YES	VB: Indicates whether any of the "Find" methods were successful in locating a record.
RecordCount	YES	YES	
Session	YES	N/A	OO: Returns the associated Session object.
SQL	YES	N/A	OO: The SQL statement used to create the dynaset.
Sort	NO	YES 3	OO: Use an ORDER BY clause.
Transactions	YES	YES	OO: Dynasets ALWAYS support Transactions.
Updatable	YES	YES	

Field Properties	**OO**	**VB**	**Comments**
Attributes	NO	YES	OO: This is part of the Data Dictionary.
CollatingOrder	NO	YES	OO: See the 'Alter Session' NLS_SORT option of the Oracle database.
Name	YES	YES	
OrdinalPosition	NO	YES	
Size	YES	YES	
SourceField	NO	YES	OO: You must examine your SQL statement.
SourceTable	NO	YES	OO: You must examine your SQL statement.
Truncated	YES	NO	OO: Indicates a LONG or LONG RAW was truncated on fetch. This indicates that you need to use GetChunk to retrieve the complete data.
Type	YES	YES	
Value	YES	YES	

Field Methods	**OO**	**VB**	**Comments**
AppendChunk	YES	YES	
FieldSize	YES	YES	OO: Available for fields up to 64KB otherwise -1 is returned.
Get Chunk	YES	YES	

Data Control Properties	**OO**	**VB**	**Comments**
AllowMoveLast	YES	NO	OO: Allow user to press MoveLast button of the data control?
AutoBinding	YES	N/A	OO: Allow automatic binding of database Parameters?
BackColor	YES	YES	
Caption	YES	YES	
Connect	YES	YES	
Database	YES	YES	
DatabaseName	YES	YES	
DragIcon	YES	YES	
DragMode	YES	YES	
EditMode	YES	YES	
Enabled	YES	YES	
Exclusive	N/A	YES	OO: Clients generally have no control over database access.
FontBold	YES	YES	
FontItalics	YES	YES	
FontName	YES	YES	
FontSize	YES	YES	
FontStrikeThru	YES	YES	
FontUnderline	YES	YES	
ForeColor	YES	YES	
Height	YES	YES	
HelpContextID	YES	NO	OO: Help target number to jump to when F1 is pressed and the data control has focus (in a running application).
Index	YES	YES	
Left	YES	YES	
MousePointer	YES	YES	
Name	YES	YES	
Options	YES	YES	The Options flags for Oracle and VB are significantly different. Please consult the respective documentation

			for details.
ReadOnly	YES	YES	
RecordSet	YES	YES	
RecordSource	YES	YES	OO: The SQL statement may be any arbitrary SELECT statement, including joins, references to views, nested selects, or remote database references.
Session	YES	N/A	OO: Returns the associated Session object.
Tag	YES	YES	
Top	YES	YES	
TrailingBlanks	YES	NO	OO: Strip trailing blanks from text retrieved from the database?
Visible	YES	YES	
Width	YES	YES	

<u>Data Control Methods</u>	**OO**	**VB**	**Comments**
Drag	YES	YES	
Move	YES	YES	
Refresh	YES	YES	
UpdateControls	NO	YES	OO: Implemented as a dynaset method, due to a VB limitation.
UpdateRecord	NO	YES	OO: Implemented as a dynaset method, due to a VB limitation.
ZOrder	YES	YES	

<u>Data Control Events</u>	**OO**	**VB**	**Comments**
DragDrop	YES	YES	
DragOver	YES	YES	
Error	YES	YES	
MouseDown	YES	YES	
MouseMove	YES	YES	
MouseUp	YES	YES	
Reposition	YES	YES	
Validate	YES	YES	

1: These Oracle Objects methods can be prefixed by "Db" if necessary since many of these names are reserved words in Visual Basic 3.0 and Access 2.0.
2: Visual Basic documentation suggests that this method should not be used.
3: Only available with the Visual Basic Professional Edition.

See Also

Method And Property Name Conflicts
OraConnection Object
OraClient Object
OraDatabase Object
OraDynaset Object
OraField Object
OraParameter Object
OraSession Object

OLE Server

See Also Objects Methods Properties

Description

Implements, using OLE 2.0, the various Oracle objects and their programmatic interfaces.

Remarks

The *Oracle Object Server* is an OLE In Process server that supports a collection of programmable objects for Oracle7 databases running either locally or remotely.   An OLE In Process supports the OLE Automation interface, has no user interface and is not embeddable.

See Also

Getting Started
Overview
Requirements

Objects

OraClient
OraConnection
OraDatabase
OraDynaset
OraField
OraParameter
OraSession

OraConnections Collection
OraFields Collection
OraParameters Collection
OraSessions Collection

OLE Server Objects

OraClient
OraConnection
OraDatabase
OraDynaset
OraField
OraParameter
OraSession

OraConnections Collection
OraFields Collection
OraParameters Collection
OraSessions Collection

OLE Server Objects

OraClient
OraConnection
OraDatabase
OraDynaset
OraField
OraParameter
OraSession

OraConnections Collection
OraFields Collection
OraParameters Collection
OraSessions Collection

OraClient Object

See Also Properties Methods

Description

An **OraClient** object defines a workstation domain, and all of that workstations **OraSession** objects are listed in the **OraClient** object's **OraSessions** collection.

Remarks

Only one **OraClient** object can exist for each workstation, and it is created automatically by the system when it is needed.

See Also

OraSession Object
OraSessions Collection

Properties

Name
Sessions

Methods

CreateSession

OraConnection Object

See Also Properties Methods

Description

An **OraConnection** object represents a single connection to an Oracle database.

Remarks

An **OraConnection** object is created automatically whenever an **OraDatabase** object is instantiated within the session, and is destroyed automatically whenever all databases using the connection are discarded. Currently there is no way to create an **OraConnection** object explicitly, only by creating an **OraDatabase** object that requires a connection.

See Also

OraConnections Collection
OraDatabase Object

Properties

- **Connect**
- **DatabaseName**
- **Session**

Methods

- **CommitTrans**
- **Rollback**

OraDatabase Object

See Also Properties Methods

Description

An **OraDatabase** object represents a single virtual login to an Oracle database.

Remarks

The **OraDatabase** object is created using a username, password, and database name combination that establishes access and security. An **OraDatabase** object can establish a connection if one does not exist, or share an existing connection if one does exist. A connection is shared when the current **Connection** and **DatabaseName** properties of an **OraDatabase** object exactly match those properties of an existing **OraDatabase** object.

See Also

Connection Property
DatabaseName Property
OpenDatabase Method.

Properties

- Connect
- Connection
- DatabaseName
- LastServerErr
- LastServerErrPos
- LastServerErrText
- Options
- Parameters
- Session

Methods

CreateDynaset
ExecuteSQL
LastServerErrReset

OraDynaset Object

See Also Properties Methods

Description

An **OraDynaset** object permits browsing and updating of data created from a SQL SELECT statement.

Remarks

The **OraDynaset** object can be thought of as a *cursor*, although in actuality several real cursors may be used to implement the **OraDynaset**'s semantics. An **OraDynaset** automatically maintains a local cache of data fetched from the server and transparently implements scrollable cursors within the browse data. Large queries may require significant local disk space; application implementors are encouraged to refine queries to limit disk usage. Placement (temporary file storage) and tuning of the local cache is accomplished through the ORAOLE.INI file

This object provides transparent mirroring of database operations, such as updates. When data is updated via the **Update** method, the local mirror image of the query is updated so that the data appears to have been changed without reevaluating the query. The same procedure is used automatically when records are added to the dynaset. Integrity checking is performed to ensure that the mirrored image of the data always matches the actual data present on the Oracle database. This integrity checking is performed only when necessary (such as just before updates occur).

During create and refresh, **OraDynaset** objects automatically bind all relevant, enabled, input parameters to the specified SQL statement, using the parameter names as placeholders in the SQL statement. This can simplify dynamic query building and increase the efficiency of multiple queries using the same SQL statement with varying WHERE clauses.

See Also

- **CreateDynaset** Method
- **ORAOLE.INI** File
- **OraParameter** Object
- **Update** Method

Properties

- BOF
- Bookmark
- Bookmarkable
- Connection
- Database
- EditMode
- EOF
- Fields
- LastModified
- Options
- RecordCount
- Session
- SQL
- Transactions
- Updatable

Methods

- **AddNew**
- **Clone**
- **Delete**
- **Edit**
- **MoveFirst**
- **MoveLast**
- **MoveNext**
- **MovePrevious**
- **Refresh**
- **Update**

OraField Object

See Also Properties Methods

Description

An OraField object represents a single column or data item within a row of a dynaset.

Remarks

An **OraField** object is accessed indirectly by retrieving a field from the **OraFields** collection of an **OraDynaset** object.

If the current row is being updated, then the **OraField** object represents the currently updated value, although the value may not yet have been committed to the database.
Assignment to the **Value** property of a field is permitted only if a record is being edited (using **Edit**) or a new record is being added (using **AddNew**). Other attempts to assign data to a field's **Value** property results in an error.

See Also

AddNew Method
Edit Method
OraDynaset Object
OraFields Collection
Value Property

Properties

- **Name**
- **Size**
- **Truncated**
- **Type**
- **Value**

Methods

- **AppendChunk**
- **FieldSize**
- **GetChunk**

OraParameter Object

See Also　　Properties　　Methods

Description

An **OraParameter** object represents a bind variable in a SQL statement or PL/SQL block.

Remarks

OraParameter objects are created, accessed, and removed indirectly through the **OraParameters** collection of an **OraDatabase** object.   Each parameter has an identifying name and an associated value. You can automatically bind a parameter to SQL and PL/SQL statements of other objects (as noted in the objects descriptions), by using the parameters name as a placeholder in the SQL or PL/SQL statement.   Such use of parameters can simplify dynamic queries and increase program performance.

Parameters are bound to SQL statements and PL/SQL blocks before execution. In the case of a SQL SELECT statement, binding occurs before dynaset creation.

The **OraParameters** collection is part of the **OraDatabase** object so that all parameters are available to any SQL statement or PL/SQL block executed within the database (via **CreateDynaset** or **ExecuteSQL**).   Before a SQL statement or PL/SQL block is executed an attempt is made to bind all parameters of the associated **OraDatabase** object.   The bindings that fail (because the parameter doesn't apply to that particular SQL statement or PL/SQL block), are noted and no attempt is made to rebind them if the SQL statement or PL/SQL block is re-executed but doesn't change.

Since neither SQL statements nor PL/SQL blocks are parsed locally (all parsing is done by Oracle), any unnecessary binding results in performance degradation.   To prevent unnecessary parameter binding, make use of the **AutoBindDisable** and **AutoBindEnable** methods.

See Also

AutoBindDisable Method
AutoBindEnable Method
CreateDynaset Method
ExecuteSQL Method
Name Property
OraDatabase Object
OraParameters Collection
Value Property

Properties

- **MinimumSize**
- **Name**
- **ServerType**
- **Status**
- **Type**
- **Value**

Methods

AutoBindDisable
AutoBindEnable

OraSession Object

See Also Properties Methods

Description

An **OraSession** object manages collections of **OraDatabase**, **OraConnection**, and **OraDynaset** objects used within an application.

Remarks

Typically, a single **OraSession** object is created for each application, but you can create named **OraSession** objects for shared use within and between applications.

The OraSession object is the top-most level object for an application. It is the only object created by the **CreateObject** VB/VBA API and not by an Oracle Objects for OLE method. The following code fragment shows how to create an **OraSession** object.

```
Declare OraSession as Object
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
```

See Also

OraClient Object
OraConnection Object
OraDatabase Object
OraDynaset Object

Properties

- Client
- Connections
- LastServerErr
- LastServerErrText
- Name
- OIPVersionNumber

Methods

- BeginTrans
- CommitTrans
- ConnectSession
- CreateNamedSession
- OpenDatabase
- LastServerErrReset
- ResetTrans
- Rollback

OraConnections Collection

See Also	Properties

Description

The **OraConnections** collection maintains a list of **OraConnection** objects. The list is not modifiable; you cannot add to or remove from this collection.

Remarks

You can access the **OraConnection** objects in this collection by subscripting (using ordinal integers) or by using the name the object was given at its creation.   You can obtain the number of **OraConnection** objects in the collection by using the **Count** property.   Referencing a subscript that does not lie within the collection (0 to Count-1) results in the return of a NULL **OraConnection** object.

See Also

Count Property
OraConnection Object

Properties

Count

OraFields Collection

See Also   Properties

Description

The **OraFields** collection maintains a list of **OraField** objects.   The list is not modifiable; you cannot add to or remove from this collection.

Remarks

You can access the **OraField** objects in this collection by subscripting (using ordinal integers) or by using the name the object was given at its creation. You can obtain the number of **OraField** objects in the collection by using the **Count** property.   Referencing a subscript that does not lie within the collection (0 to Count-1) results in the return of a NULL **OraField** object.

See Also

OraField Object

Properties

Count

OraParameters Collection

See Also Properties Methods

Description

The **OraParameters** collection maintains a list of **OraParameter** objects. Unlike the other collection objects, this list is modifiable; you can add to and remove from the collection.

Remarks

You can access the **OraParameter** objects in this collection by subscripting (using ordinal integers) or by using the name the object was given at its creation. You can obtain the number of **OraParameter** objects in the collection by using the **Count** property. Referencing a subscript that does not lie within the collection (0 to Count-1) results in the return of a NULL **OraParameter** object.

In addition to accessing the **OraParameter** objects of the collection, you can use the collection to create and destroy parameters by using the **Add** and **Remove** methods, respectively.

See Also

Add Method
Count Property
OraParameter Object
Remove Method

Properties

Count

Methods

- Add
- Remove

OraSessions Collection

See Also Properties

Description

The **OraSessions** collection maintains a list of **OraSession** objects. The list is not modifiable; you cannot add to or remove from this collection.

Remarks

You can access the **OraSession** objects in this collection by subscripting (using ordinal integers) or by using the name the object was given at its creation. You can obtain the number of **OraSession** objects in the collection by using the **Count** property. Referencing a subscript that does not lie within the collection (0 to Count-1) results in the return of a NULL **OraSession** object.

See Also

Count Property
OraSession Object

Properties

Count

Properties

BOF	MinimumSize
Bookmark	Name
Bookmarkable	OIPVersionNumber
Client	Options
Connect	Parameters
Connection	RecordCount
Count	ServerType
Database	Session
DatabaseName	Sessions
EditMode	Size
EOF	Sort
Fields	SQL
Filter	Status
LastModified	Transactions
LastServerErr	Truncated
LastServerErrPos	Type
LastServerErrText	Updatable
	Value

OLE Server Properties

- BOF
- Bookmark
- Bookmarkable
- Client
- Connect
- Connection
- Count
- Database
- DatabaseName
- EditMode
- EOF
- Fields
- Filter
- LastModified
- LastServerErr
- LastServerErrPos
- LastServerErrText
- MinimumSize
- Name
- OIPVersionNumber
- Options
- Parameters
- RecordCount
- ServerType
- Session
- Sessions
- Size
- Sort
- SQL
- Status
- Transactions
- Truncated
- Type
- Updatable
- Value

OLE Server Properties

- BOF
- Bookmark
- Bookmarkable
- Client
- Connect
- Connection
- Count
- Database
- DatabaseName
- EditMode
- EOF
- Fields
- Filter
- LastModified
- LastServerErr
- LastServerErrPos
- LastServerErrText
- MinimumSize
- Name
- OIPVersionNumber
- Options
- Parameter
- RecordCount
- ServerType
- Session
- Sessions
- Size
- Sort
- SQL
- Status
- Transactions
- Truncated
- Type
- Updatable
- Value

BOF Property

See Also Example

Applies To

OraDynaset Object

Description

Returns whether the current record position in a dynaset is before the first record.   Not available at design time and read-only at run time.

Usage

bof_status = *oradynaset*.**BOF**

Remarks

Returns **True** if an attempt has been made to move before the first record in the dynaset using **MovePrevious**.   Returns **False** otherwise.

If a recordset is empty, both **BOF** and **EOF** return **True**.

Data Type

Integer (Boolean)

See Also

EOF Property
MoveFirst Method
MovePrevious Method
OraDynaset Object

BOF Property Example

This example demonstrates the use of the BOF and EOF properties to detect the limits of a recordset. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As object
 Dim OraDatabase As object
 Dim OraDynaset As object

 'Create the OraSession Object
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select empno, ename from emp",
0&)

 'Traverse until EOF is reached
 Do Until OraDynaset.EOF
  OraDynaset.DbMoveNext
  Loop
 MsgBox "Reached EOF"

 'When EOF is True there is no current record.
 'The current recordset position is now AFTER
 'the last record.
 OraDynaset.DbMoveLast

 Do Until OraDynaset.BOF
  OraDynaset.DbMovePrevious
 Loop
 MsgBox "Reached BOF"

 'When BOF is True there is no current record.
 'The current recordset position is now BEFORE
 'AFTER the last record.

 OraDynaset.DbMoveFirst
 'The recordset is now positioned at the first
 'record.

End Sub
```

Bookmark Property

See Also Example

Applies To

OraDynaset Object

Description

Determines the current record of a recordset. Not available at design time and read/write at run time.

Usage

row_bookmark = ora*dynaset*.**Bookmark**
oradynaset.**Bookmark** = *row_bookmark*

Remarks

The first form returns a bookmark for the current row. The second form repositions the bookmark to refer to a specific record within the dynaset.

Bookmarks exist only for the life of the dynaset and are specific to a particular dynaset (they cannot be shared among dynasets). (Note, however, that bookmarks of a dynaset and its clone are interchangeable.)

Before attempting to use bookmarks, check the **Bookmarkable** property of that dynaset to see if it supports bookmarks.

Data Type

The value is a string of binary data, but can be stored in a variable of **String** or **Variant** data type. The length of the string cannot be predicted, so do not use a fixed-length string.

See Also

Bookmarkable Property
Clone Method
LastModified Property
OraDynaset Object

Bookmark Property Example

This example demonstrates the use of the **Bookmark** property to return to a previously known record quickly.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraClient As Object
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object
 Dim Bookmark2 As String

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Get the client object.
 Set OraClient = OraSession.Client

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Move to the second record and display empno.
 OraDynaset.DbMoveNext
 MsgBox "Second Record, Employee #" & OraDynaset.Fields("EMPNO").value
 Bookmark2 = OraDynaset.Bookmark

 'Move to the last record and display empno.
 OraDynaset.DbMoveLast
 MsgBox "Last Record, Employee #" & OraDynaset.Fields("EMPNO").value

 'Move back to the second record using the bookmark.
 OraDynaset.Bookmark = Bookmark2
 MsgBox "Second Record, Employee #" & OraDynaset.Fields("EMPNO").value

End Sub
```

BookMarkable Property

See Also

Applies To

OraDynaset Object

Description

Indicates whether the specified dynaset can supports bookmarks.   Not available at design time and read-only at run time.

Usage

if_bookmarkable = *oradynaset***.Bookmarkable**

Remarks

This property returns **True** unless the No Cache mode was set when the specified dynaset was created, in which case it returns **False**.

Data Type

Integer (Boolean)

See Also

BookMark Property
CreateDynaset Method
OraDynaset Object

Client Property

See Also

Applies To

OraSession Object

Description

Returns the OraClient object associated with the specified session.   Not available at design time and read-only at run time.

Usage

Set *oraclient* = *orasession***.Client**

Remarks

Each machine has only one client object, so this property returns the same object for all sessions on the same workstation.

Data Type

OLE Object (**OraClient**)

See Also

OraClient Object
OraSession Object

Connect Property

See Also Example

Applies To

OraConnection Object, **OraDatabase** Object

Description

Returns the username of the connect string associated with the connection. Not available at design time and read-only at run time.

Usage

connect_string = *oraconnection*.**Connect**
connect_string = *oradatabase*.**Connect**

Remarks

OraConnection.**Connect** returns the username of the connect string associated with the connection.

OraDatabase.**Connect** returns the username of the connect string associated with the specified database. It is equivalent to referencing *OraDatabase*.**Connection.Connect**.

The password associated with the username is never returned.

Data Type

String

See Also

OpenDatabase Method
OraConnection Object
OraDatabase Object
DatabaseName Property

Connect Property Example

This example demonstrates the use of the **Connect** and **DatabaseName** properties to display the username and database name that have been connected to.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Display the username and database to which we are connected.
 MsgBox "Connected to " & OraDatabase.Connect & "@" & OraDatabase.DatabaseName

End Sub
```

Connection Property

See Also

Applies To

OraDatabase Object, **OraDynaset** Object

Description

Returns the **OraConnection** object associated with the specified database or dynaset.   Not available at design time and read-only at run time.

Usage

Set *oraconnection* = *oradatabase*.**Connection**
Set *oraconnection* = *oradynaset*.**Connection**

Remarks

OraDatabase.**Connection** returns the connection object associated with the specified database.   Each database is associated with one connection object, but many databases can share the same connection object.

OraDynaset.**Connection** returns the connection object associated with this dynaset.   It is equivalent to referencing *oradynaset*.**Database.Connection**.

Data Type

OLE Object (**OraConnection**)

See Also

OraConnection Object
OraConnections Collection
OraDatabase Object
OraDynaset Object

Connections Property

See Also

Applies To

OraSession Object

Description

Returns the **OraConnections** collection of the specified session.   Not available at design time and read-only at run time.

Usage

Set *oraconnections_collection* = *orasession***.Connections**

Remarks

You can access the connections in this collection by subscripting (using ordinal integer numbers).   You can obtain the number of connections in the collection using the **Count** property of the returned collection.   Integer subscripts begin with 0 and end with **Count** - 1.   Out-of-range indices and invalid names return a NULL **OraConnection** object.

Data Type

OLE Object (**OraParameters**)

See Also

CountProperty
OraConnection Object
OraConnections Collection
OraSession Object

Count Property

See Also Example

Applies To

OraConnections Collection, **OraFields** Collection, **OraParameters** Collection, **OraSessions** Collection.

Description

Returns the number of objects in the specified collection. Not available at design time and read-only at run time.

Usage

connection_count = *oraconnections*.**Count**
field_count = *orafields*.**Count**
parameter_count = *oraparameters*.**Count**
session_count = *orasessions*.**Count**

Remarks

Use this property to determine the number of objects in the specified collection.

Data Type

Integer

See Also

OraConnection Object
OraConnections Collection
OraField Object
OraFields Collection
OraSession Object
OraSessions Collections

Count Property Example

This example demonstrates the use of the **Count** property to display the number of objects in the **OraSessions**, **OraConnections**, and **OraFields** Collections.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraClient As Object
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Get the client object.
 Set OraClient = OraSession.Client

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 MsgBox "You have " & OraClient.Sessions.Count & " OraSession Object(s)."
 MsgBox "You have " & OraSession.Connections.Count & " OraConnection 
Object(s)."
 MsgBox "You have " & OraDynaset.Fields.Count & " OraField Object(s)."

End Sub
```

Database Property

See Also

Applies To

OraDynaset Object

Description

Returns the OraDatabase object associated with the specified dynaset.   Not available at design time and read-only at run time.

Usage

Set *oradatabase* = *oradynaset***.Database**

Remarks

This property returns the **OraDatabase** object from which the specified dynaset was created.

Data Type

OLE Object (**OraDatabase**)

See Also

CreateDynaset Method
OraDatabase Object
OraDynaset Object

DatabaseName Property

See Also Example

Applies To

OraConnection Object, **OraDatabase** Object

Description

Returns the name of the database associated with the specified object.   Not available at design time and read-only at run time.

Usage

database_name = *oraconnection*.**DatabaseName**
database_name = *oradatabase*.**DatabaseName**

Remarks

oraconnection.**DatabaseName** returns the name of the database, as specified in the **OpenDatabase** method.

oradatabase.**DatabaseName** returns the database name associated with the connection.   It is the same as the referencing *oradatabase*.**Connection.DatabaseName**.

Data Type

String

See Also

Connect Property
OpenDatabase Method
OraConnection Object
OraDatabase Object

DatabaseName Property Example

This example demonstrates the use of the **Connect** and **DatabaseName** properties to display the username and database to which you have connected. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Display the username and database to which you have connected.
 MsgBox "Connected to " & OraDatabase.Connect & "@" & OraDatabase.DatabaseName

End Sub
```

EditMode Property

See Also

Applies To

OraDynaset Object

Description

Returns the current editing state for the current row.   Not available at design time and read-only at run time.

Usage

edit_mode = *oradynaset*.**EditMode**

Remarks

The **EditMode** property values are:

Constant	Value	Description
ORADATA_EDITNONE	0	No editing in progress.
ORADATA_EDITMODE	1	Editing is in progress on an existing row.
ORADATA_EDITADD	2	A new record is being added and the copy buffer does not currently represent an actual row in the database.

These values are located in the file **ORACONST.TXT** and are intended to match similar constants in the Visual Basic file **CONSTANT.TXT**.

This property is affected only by the **Edit**, **AddNew**, and **Update** methods.

Data Type

Integer

See Also

AddNew Method
Edit Method
OraDynaset Object
Update Method

EOF Property

See Also Example

Applies To

OraDynaset Object

Description

Returns whether the current record position in a dynaset is after the last record. Not available at design time and read-only at run time.

Usage

eof_status = *oradynaset*.**EOF**

Remarks

Returns **True** if an attempt has been made to move after the last record in the dynaset using **MoveNext**. Returns **False** otherwise.

If a recordset is empty, both **BOF** and **EOF** return **True**.

Data Type

Integer (Boolean)

See Also

- **BOF** Property
- **MoveLast** Method
- **MoveNext** Method
- **OraDynaset** Object

EOF Property Example

This example demonstrates the use of the **BOF** and **EOF** properties to detect the limits of a recordset. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As object
 Dim OraDatabase As object
 Dim OraDynaset As object

 'Create the OraSession Object
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object
 Set OraDynaset = OraDatabase.DbCreateDynaset("select empno, ename from emp",
0&)

 'Traverse until EOF is reached
 Do Until OraDynaset.EOF
  OraDynaset.DbMoveNext
  Loop
 MsgBox "Reached EOF"

 'When EOF is True there is no current record.
 'The current recordset position is now AFTER
 'the last record.
 OraDynaset.DbMoveLast

 Do Until OraDynaset.BOF
  OraDynaset.DbMovePrevious
 Loop
 MsgBox "Reached BOF"

 'When BOF is True there is no current record.
 'The current recordset position is now BEFORE
 'AFTER the last record.

 OraDynaset.DbMoveFirst
 'The recordset is now positioned at the first
 'record.

End Sub
```

Fields Property

See Also

Applies To

OraDynaset Object

Description

Returns the collection of fields for the current row.   Not available at design time and read-only at run time.

Usage

Set *orafields_collection* = *oradynaset***.Fields**

Remarks

You can access the fields in this collection by subscripting (using ordinal integer numbers) or by using a string denoting the field (column) name.   You can obtain the count of the number of fields using the **Count** property on the returned collection.   A subscript that does not lie within the collection (0 to Count - 1) results in the return of a NULL **OraField** object.

Data Type

OLE Object (**OraFields**)

See Also

Count Property
OraDynaset Object

Filter Property

See Also

Remarks

The **OraDynaset** object does not support this property.   Refine your record selection by using a SQL WHERE clause or by using SQL parameters.

See Also

- **CreateDynaset** Method
- **OraDynaset** Object
- **OraFields** Collection
- **OraParameter** Object

LastModified Property

See Also

Applies To

OraDynaset Object

Description

Returns the bookmark of the row that was last modified by an **Edit** or an **AddNew** operation.   Not available at design time and read-only at run time.

Usage

last_modified_bookmark = *oradynaset***.LastModified**

Remarks

Use this property to make the last modified record the current record.

Data Type

The value is a string of binary data, but can be stored in a variable of **String** or **Variant** data type.   The length of the string cannot be predicted, so do not use a fixed-length string.

See Also

AddNew Method
BookMark Property
Edit Method

LastServerErr Property

See Also Example

Applies To

OraDatabase Object, **OraSession** Object

Description

Returns the last non-zero error code generated by an Oracle database function for the specified object. Not available at design time and read-only at run time.

Usage

error_number = *oradatabase***.LastServerErr**
error_number = *orasession***.LastServerErr**

Remarks

This property represents the last non-zero return value from an Oracle Call Interface (OCI) database function, or 0 if no error has occurred since the last **LastServerErrReset** request. For efficiency, only non-zero return values are returned; therefore, a non-zero value does not necessarily indicate that the most recently called OCI database function generated the error (because zero return values are not returned by way of **LastServerErr**).

Orasession.**LastServerErr** returns all errors related to connections, such as errors on **OpenDatabase**, **BeginTrans**, **CommitTrans**, **Rollback and ResetTrans**.

Oradatabase.**LastServerErr** returns all errors related to an Oracle cursor, such as errors on dynasets and from **ExecuteSQL**.

Data Type

Long Integer

See Also

ExecuteSQL Method
LastServerErrReset Method
LastServerErrText Property
OpenDatabase Method
OraDatabase Object
OraSession Object

LastServerErr Property Example

This example demonstrates the use of the **LastServerErr** and **LastServerErrText** properties to determine whether an Oracle error has occurred using **CreateDynaset** and to display the error message, respectively. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Set up an error handler.
 On Error GoTo errhandler

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Attempt to Create the OraDynaset Object.
 'Notice that the FROM keyword is missing from the SQL statement.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * emp", 0&)

Exit Sub

errhandler:

 'Check to see if an Oracle error has occurred.
 If OraDatabase.LastServerErr <> 0 Then
  MsgBox OraDatabase.LastServerErrText
 Else 'Must be some non-Oracle error
  MsgBox "VB:" & Err & " " & Error(Err)
 End If

 Exit Sub

End Sub
```

LastServerErrPos Property

See Also

Applies To

OraDatabase Object

Description

Returns the position in a SQL statement at which a parse error occurred. Not available at design time and read-only at run time.

Usage

error_pos = *oradatabase***.LastServerErrPos**

Remarks

LastServerErrPos returns 0 if no SQL statements have been parsed, -1 if the last parse was successful, and >= 0 if the last parse failed. Parsing is done on SQL statements before execution (using **CreateDynaset** or **ExecuteSQL**).

Data Type

Integer

See Also

CreateDynaset Method
ExecuteSQL Method
LastServerErr Property
LastServerErrText Property
OraDatabase Object

LastServerErrText Property

See Also Example

Applies To

OraDatabase Object, **OraSession** Object

Description

Returns the textual message associated with the current **LastServerErr** of the specified object. Not available at design time and read-only at run time.

Usage

error_text = *orasession*.**LastServerErrText**
error_text = *oradatabase*.**LastServerErrText**

Remarks

The returned value indicates one of three possible states:

1. If NULL is returned, an Oracle Call Interface (OCI) database function has not returned an error since the most recent **LastServerErrReset**.
2. If a non-NULL value is returned, an OCI function has returned an error code; the returned string is the associated message.
3. If the message is empty, then additional information was not available.

Data Type

String

See Also

LastServerErr Property
LastServerErrReset Method
OraDatabase Object
OraSession Object

LastServerErrText Property Example

This example demonstrates the use of the **LastServerErr** and **LastServerErrText** properties to determine whether an Oracle error has occurred using **CreateDynaset** and to display the error message, respectively.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Set up an error handler.
 On Error GoTo errhandler

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Attempt to Create the OraDynaset Object.
 'Notice that the FROM keyword is missing from the SQL statement.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * emp", 0&)

Exit Sub

errhandler:


 'Check to see if an Oracle error has occurred.
 If OraDatabase.LastServerErr <> 0 Then
  MsgBox OraDatabase.LastServerErrText
 Else 'Must be some non-Oracle error.
  MsgBox "VB:" & Err & " " & Error(Err)
 End If

 Exit Sub

End Sub
```

MinimumSize Property

See Also

Applies To

OraParameter Object

Description

Specifies the minimum size of an **OraParameter** string buffer.   Read/write at design time and run time.

Usage

*oraparameter***.MinimumSize** = *data_width*

Remarks

Use this property to specify the minimum number of characters or bytes to be allocated for the buffer used to bind parameters with the **ExecuteSQL** method. The contents of the parameter remain unchanged.

Data Type

Integer

See Also

Add Method
ExecuteSQL Method
OraParameter Object

Name Property

See Also

Applies To

OraClient Object, **OraField** Object, **OraParameter** Object, **OraSession** Object

Description

Returns the name used to identify the given object.   Not available at design time and read-only at run time.

Usage

client_name = *oraclient*.**Name**
field_name = *orafield*.**Name**
parameter_name = *oraparameter*.**Name**
session_name = *orasession*.**Name**

Remarks

oraclient.**Name** returns the name of the specified **OraClient** object.   This value is always "<local>".

orafield.**Name** returns the name of the specified **OraField** object.   If this is a true database field (not an alias), this usage returns the name of the field as it appears on the server.   If a SQL statement was executed that contains, for example, calculated select list items or column aliases, then the name is the actual text provided in the SELECT statement.

oraparameter.**Name** returns the name of the specified **OraParameter** object. In addition to identifying the parameter within a parameters collection, the parameter name is also used to match placeholders within SQL and PL/SQL statements for the purposes of parameter binding.

orasession.**Name** returns the name of the specified **OraSession** object.   For automatically created sessions, this is the name assigned by the system (usually a hexadecimal number).   For user-created sessions, this is the name originally provided in the **CreateSession** method.   Once created, a session name cannot be changed.

Data Type

String

See Also

CreateSession Method
OraClient Object
OraField Object
OraParameter Object
OraSession Object

OIPVersionNumber Property

See Also

Applies To

OraSession Object

Description

Returns the version number of the Oracle Object Server.   Not available at design time and read-only at run time.

Usage

version_number = *orasession***.OIPVersionNumber**

Remarks

This property returns a unique identifier for each release of the Oracle Object Server.

Data Type

String

See Also

OraSession Object

Options Property

See Also

Applies To

OraDatabase Object, **OraDynaset** Object

Description

Returns the options flag originally passed to the specified object. Not available at design time and read-only at run time.

Usage

options = *oradatabase***.Options**
options = *oradynaset***.Options**

Remarks

Refer to the **OpenDatabase** method for a description of the possible values of *oradatabase***.Options**.
Refer to the **CreateDynaset** method for a description of the possible values of *oradynaset***.Options**.

Data Type

Long Integer

See Also

CreateDynaset Method
OpenDatabase Method
OraDatabase Object
OraDynaset Object

Parameters Property

See Also

Applies To

OraDatabase Object

Description

Returns the **OraParameters** collection of the specified database. Not available at design time and read-only at run time.

Usage

Set *oraparameters_collection* = *oradatabase***.Parameters**

Remarks

You can access the parameters in this collection by subscripting (using ordinal integer numbers) or by using the name the parameter was given at its creation. You can obtain the number of parameters in the collection using the **Count** property of the returned collection. Integer subscripts begin with 0 and end with Count-1. Out-of-range indices and invalid names return a NULL **OraParameter** object.

In addition to accessing the parameters of the collection, you can also use the collection to create and destroy parameters using the **Add** and **Remove** methods, respectively.

Data Type

OLE Object (**OraParameters**)

See Also

Add Method
Count Property
OraParameter Object
OraParameters Collection
Remove Method

RecordCount Property

See Also Example

Applies To

OraDynaset Object

Description

Returns the count of the total number of records in the dynaset. Not available at design time and read-only at run time.

Usage

record_count = *oradynaset*.**RecordCount**

Remarks

Referencing this property requires that the entire result table be fetched immediately from Oracle in order to determine the count of records. While it would have been possible to execute a separate query (using COUNT(*)), the consistency of the count could not be ensured since row locking is not done to ensure repeatable reads.

Referencing this property while using the ORADYN_NOCACHE option of the **CreateDynaset** method causes an implicit **MoveLast** and makes the current record the last record in the dynaset.

Data Type

Long Integer

See Also

CreateDynaset Method
Locks and Editing
MoveLast Method
OraDynaset Object

RecordCount Property Example

This example demonstrates the use of the **RecordCount** property to determine the number of records retrieved.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the dynaset.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Display the number of records. Note that this causes
 'all records to be fetched to ensure an accurate count.

 MsgBox OraDynaset.RecordCount & " records retrieved."

End Sub
```

ServerType Property

See Also Example

Applies To

OraParameter Object

Description

Specifies the Oracle external type of SQL or PL/SQL bind variable. Read/write at design time and run time.

Usage

oraparamter.**ServerType** = oracle_type

Remarks

Used to specify the external data type of SQL or PL/SQL (in/out) bind variables. This is necessary since no local parsing of the SQL statement or PL/SQL block is done to match the data types of placeholders in the SQL statement or PL/SQL block.

The following Oracle external data types are supported.

Constant	Value	External Data Type
ORATYPE_VARCHAR2	1	VARCHAR2
ORATYPE_NUMBER	2	NUMBER
ORATYPE_SINT	3	SIGNED INTEGER
ORATYPE_FLOAT	4	FLOAT
ORATYPE_STRING	5	Null Terminated STRING
ORATYPE_VARCHAR	9	VARCHAR
ORATYPE_DATE	12	DATE
ORATYPE_UINT	68	UNSIGNED INTEGER
ORATYPE_CHAR	96	CHAR
ORATYPE_CHARZ	97	Null Terminated CHAR

These values can be found in the file **ORACONST.TXT**.

Data Type

Integer

See Also

Add Method
OraParameter Object

ServerType Property Example

This example demonstrates the **Add** and **Remove** parameter methods, the ServerType parameter property and the the **ExecuteSQL** database method to call a Stored Procedure and Function (located in ORAEXAMP.SQL).   Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object.
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add EMPNO as an Input/Output parameter and set its initial value.
 OraDatabase.Parameters.Add "EMPNO", 7369, ORAPARM_INPUT
 OraDatabase.Parameters("EMPNO").ServerType = ORATYPE_NUMBER

 'Add ENAME as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "ENAME", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("ENAME").ServerType = ORATYPE_VARCHAR2

 'Add SAL as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "SAL", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("SAL").ServerType = ORATYPE_NUMBER

 'Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME.
 ' This Stored Procedure can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")
 'Display the employee number and name.

 'Execute the Stored Function Employee.GetSal to retrieve SAL.
 ' This Stored Function can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("declare SAL number(7,2);
Begin :SAL:=Employee.GetEmpSal (:EMPNO); end;")

 'Display the employee name, number and salary.
 MsgBox "Employee " & OraDatabase.Parameters("ENAME").value & ", #" &
OraDatabase.Parameters("EMPNO").value & ",Salary=" &
OraDatabase.Parameters("SAL").value

 'Remove the Parameters.
 OraDatabase.Parameters.Remove "EMPNO"
 OraDatabase.Parameters.Remove "ENAME"
 OraDatabase.Parameters.Remove "SAL"

End Sub
```

Session Property

See Also

Applies To

OraConnection Object, **OraDatabase** Object, **OraDynaset** Object

Description

Returns the **OraSession** object associated with the specified object. Not available at design time and read-only at run time.

Usage

Set *orasession* = *oraconnection***.Session**
Set *orasession* = *oradatabase***.Session**
Set *orasession* = *oradynaset***.Session**

Remarks

oraconnection.**Session** returns the **OraSession** object in which this OraConnection object resides.

oradatabase.**Session** returns the **OraSession** object associated with this OraDatabase object. Each database is a part of one and only one session, which is by default the session associated with the application.

oradynaset.**Session** returns the **OraSession** object associated with this **OraDynaset** object.

Data Type

OLE Object (**OraSession**)

See Also

OraConnection Object
OraDatabase Object
OraDynaset Object
OraSession Object
OraSessions Collection

Sessions Property

See Also

Applies To

OraClient Object

Description

Returns the collection of all sessions for the specified **OraClient** object.   Not available at design time and read-only at run time.

Usage

Set *orasessions_collection* = *oraclient*.**Sessions**

Remarks

You can access a session in this collection by subscripting (using ordinal numbers) or by using the name the session was given at its creation. You can obtain the total number of sessions in the collection by using the **Count** property of the returned collection. Integer subscripts begin with 0 and end with Count-1. Out-of-range indices and invalid names return a NULL OraSession object.

Data Type

OLE Object (**OraSessions**)

See Also

Count Property
OraSession Object
OraSessions Collection

Size Property

See Also

Applies To

OraField Object

Description

Returns the number of characters or bytes of the variant associated with the returned value of this field.   Not available at design time and read-only at run time.

Usage

field_size = *orafield***.Size**

Remarks

This usage always returns 0 for LONG or LONG RAW fields.   Use the **FieldSize** method to determine the length of LONG or LONG RAW fields.

Data Type

Long Integer

See Also

- **OraFields** Collection
- **FieldSize** Property
- Long and Long Raw Columns
- **Type** Property

Sort Property

See Also

Remarks

The **OraDynaset** object does not support this property.   Sort your recordset by using a SQL ORDER BY clause.

See Also

CreateDynaset Method
OraDynaset Object

SQL Property

See Also Example

Applies To

OraDynaset Object

Description

Returns or sets the SQL statement used to create the specified dynaset. Read/write at design time and run time.

Usage

SQL_statement = *oradynaset***.SQL**
*oradynaset***.SQL** = *SQL_statement*

Remarks

The first form returns the contents of the SQL statement buffer.
The second form sets the contents of the SQL statement buffer.

The SQL statement buffer initially contains the SQL statement used to create the dynaset. The contents of the SQL statement buffer are executed whenever the **Refresh** method is issued.

Data Type

String

See Also

OraDynaset Object
Refresh Method

SQL Property Example

This example demonstrates the use of parameters, the **Refresh** method and the **SQL** property to restrict selected records.   Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create a parameter with an initial value.
 OraDatabase.Parameters.Add "job", "MANAGER", 1

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where
job=:job", 0&)

 'Notice that the SQL statement is NOT modified.
 MsgBox OraDynaset.SQL

 'Currently, OraDynaset only contains employees whose
 'job is MANAGER.

 'Change the value of the job parameter.
 OraDatabase.Parameters("job").Value = "SALESMAN"

 'Refresh the dynaset.
 OraDynaset.DbRefresh

 'Currently, OraDynaset only contains employees whose
 'job is SALESMAN.

 'Notice that the SQL statement is NOT modified.
 MsgBox OraDynaset.SQL

 'Remove the parameter.
 OraDatabase.Parameters.Remove ("job")

 End Sub
```

Status Property

See Also Example

Applies To

OraParameter Object

Description

Returns an integer indicating the status of the specified parameter. Not available at design time and read-only at run time.

Usage

parameter_status = *oraparameter***.Status**

Remarks

The **Status** property is interpreted as a series of bits, each providing information about the parameter. Parameters can be bound only if they are enabled, and can be enabled only if they are autoenabled.

The parameter **Status** property bit values are:

Constant	Value	Description
ORAPSTAT_INPUT	&H1&	Parameter can be used for input.
ORAPSTAT_OUTPUT	&H2&	Parameter can be used for output.
ORAPSTAT_AUTOENABLE	&H4&	Parameter is AutoBindEnabled.
ORAPSTAT_ENABLE	&H8&	Parameter is Enabled. This bit is always set.

These values are located in the file **ORACONST.TXT**.

Data Type

Integer

See Also

Add Method
AutoBindDisable Method
AutoBindEnable Method
OraParameter Object
Remove Method

Status Property Example

This example demonstrates the use of parameters and **ExecuteSQL** to call a Stored Procedure (located in ORAEXAMP.SQL). After calling the Stored Procedure, the **Status** property of each parameter is checked. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add EMPNO as an Input parameter and set its initial value.
 OraDatabase.Parameters.Add "EMPNO", 7369, ORAPARM_INPUT

 'Add ENAME as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "ENAME", 0, ORAPARM_OUTPUT

 'Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME.
 ' This Stored Procedure is located in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")

 If OraDatabase.Parameters("EMPNO").Status & ORAPSTAT_INPUT Then
  MsgBox "Parameter EMPNO used for input."
 End If

 If OraDatabase.Parameters("ENAME").Status & ORAPSTAT_OUTPUT Then
  MsgBox "Parameter ENAME used for output."
 End If

'Display the employee number and name.
 MsgBox OraDatabase.Parameters("EMPNO").value
 MsgBox OraDatabase.Parameters("ENAME").value

 'Remove the Parameters.
 OraDatabase.Parameters.Remove "EMPNO"
 OraDatabase.Parameters.Remove "ENAME"

End Sub
```

Transactions Property

See Also

Applies To

OraDynaset Object

Description

Returns whether or not the given dynaset can support transaction processing. Not available at design time and read-only at run time.

Usage

if_transactions = *oradynaset*.**Transactions**

Remarks

This property never returns **False** and has only been implemented for compatibility.

Data Type

Integer (Boolean)

See Also

- **BeginTrans** Method
- **CommitTrans** Method
- **OraDynaset** Object
- **ResetTrans** Method
- **Rollback** Method

Truncated Property

See Also

Applies To

OraField Object

Description

Returns whether or not a field value was truncated when fetched.   Not available at design time and read-only at run time.

Usage

field_status = *orafield***.Truncated**

Remarks

This property returns **True** if truncated data will be returned and **False** otherwise.   Truncation can only occur for LONG or LONG RAW fields.   Use this property to decide whether more data needs to be retrieved from Oracle using the **GetChunk** method.

Data Type

Integer (Boolean)

See Also

GetChunk Method
OraField Object
Type Property

Type Property

See Also

Applies To

OraField Object, **OraParameter** Object

Description

Returns the variant type of the specified object.   Not available at design time and read-only at run time.

Usage

data_type = *orafield*.**Type**
data_type = *oraparameter*.**Type**

Remarks

orafield.**Type** returns the variant data type (see Visual Basic documentation) associated with the returned value of this field.

Users can expect the following mapping from Oracle internal data types:

Oracle Data Type	**Constant**	**Value**	**Data Type**
CHAR	ORADB_TEXT	10	String
DATE	ORADB_DATE	8	Variant
LONG	ORADB_MEMO	12	String
LONG RAW	ORADB_LONGBINARY	11	String
NUMBER (1-4,0)	ORADB_INTEGER	3	Integer
NUMBER (5-9,0)	ORADB_LONG	4	Long Integer
NUMBER (10-15,0)	ORADB_DOUBLE	7	Double
NUMBER (16-38,0)	ORADB_TEXT	10	Text
NUMBER (1-15,n)	ORADB_DOUBLE	7	Double
NUMBER (16-38,n)	ORADB_TEXT	10	Text
RAW	ORADB_LONGBINARY	11	String
VARCHAR2	ORADB_TEXT	10	String

These values are located in the file **ORACONST.TXT** and are intended to match similar constants in the Visual Basic file **DATACONS.TXT**.

oraparameter.**Type** returns an integer indicating the variant data type that is actually bound to the SQL statement.   This may differ from the variant data type of *oraparameter*.**Value**, because internal conversions may be necessary to obtain a data type common to both Visual Basic and the Oracle database.

Note that fields of type DATE are returned in the default Visual Basic format of "MM/DD/YY" even though the default Oracle date format is "DD-MMM-YY".

Data Type

Integer

See Also

OraField Object
OraParamter Object
Value Property

Updatable Property

See Also Example

Applies To

OraDynaset Object

Description

Returns whether or not the specified dynaset is updatable.   Not available at design time and read-only at run time.

Usage

if_updatable = *oradynaset***.Updatable**

Remarks

Returns **True** if the rows in the specified dynaset can be updated and **False** otherwise.

The updatability of the resultant dynaset depends on the Oracle SQL rules of updatability, on the access you have been granted, and on the read-only flag of the **CreateDynaset** method.

In order to be updatable, three conditions must be met:

1. the SQL statement must refer to a simple column list or to the entire column list (*),
2. the SQL statement must not set the read-only flag of the options argument, and
3. Oracle must permit ROWID references to the selected rows of the query.

Any SQL statement that does not meet these criteria is processed, but the results are not updatable and this property returns **False**.

Data Type

Integer (Boolean)

See Also

CreateDynaset Method
RecordSource Property
SQL Property

Updatable Property Example

This example demonstrates the use of **Updatable**.   Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create an updatable dynaset using a simple query.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)
 Call IsDynUpdatable(OraDynaset)

 'Create a nonupdatable dynaset using column aliases.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select ename EmployeeName,
empno EmployeeNumber, sal Salary from emp", 0&)
 Call IsDynUpdatable(OraDynaset)

 'Create a nonupdatable dynaset using a join.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select ename, emp.deptno, loc
from emp,dept where emp.deptno = dept.deptno", 0&)
 Call IsDynUpdatable(OraDynaset)

End Sub


Sub IsDynUpdatable (odyn As Object)

 'Check to see if the dynaset is updatable.
 If odyn.Updatable = True Then
  MsgBox "Created an UPDATABLE dynaset from: '" & odyn.SQL & "'"
 Else
  MsgBox "Created a READ-ONLY dynaset from: '" & odyn.SQL & "'"
 End If

End Sub
```

Value Property

See Also

Applies To

OraField Object, **OraParameter** Object

Description

Returns or sets the value of the given object.   Not available at design time and read/write at run time.

Usage

*orafield***.Value** = *data_value*
data_value = *orafield***.Value**
*oraparameter***.Value** = *data_value*
data_value = *oraparameter***.Value**

Remarks

Orafield.**Value** returns the value of the field as a variant.   *data_value* = *orafield*.**Value** sets the contents of the field. The variant type reflects the data type as defined by the **Type** property.   Fields can contain NULL values.   You can test the **Value** property with the Visual Basic function IsNull() to determine whether the value is null upon return.   You can also assign NULL to the **Value** property whenever the current record is editable.   Field values are cached locally as the data is retrieved from the database.   However, in the case of a LONG or LONG RAW fields, some data may not be retrieved and stored locally.   In these cases, data is retrieved as required using the methods described in the **GetChunk** field method.   The maximum size of a LONG or LONG RAW field that can be retrieved directly through the **Value** property is approximately 64KB.   You must retrieve data fields larger than 64KB indirectly, using the **GetChunk** method.

OraParameter.**Value** returns the value of the parameter as a variant. *data_value* = *oraparameter*.**Value** sets the contents of the parameter. Note that changing the variant data type of the value can have significant impact on the processing of associated SQL and PL/SQL statements.

Note that fields of type DATE are returned in the default Visual Basic format of "MM/DD/YY" even though the default Oracle date format is "DD-MMM-YY".

Data Type

Variant

See Also

GetChunk Method
OraField Object
OraParameter Object
Type Property

Methods

Add
AddNew
AppendChunk
AutoBindDisable
AutoBindEnable
BeginTrans
Clone
CommitTrans
ConnectSession
CreateDynaset
CreateNamedSession
CreateSession
Delete
Edit
ExecuteSQL
FieldSize
GetChunk
LastServerErrReset
MoveFirst
MoveLast
MoveNext
MovePrevious
OpenDatabase
Refresh
Remove
ResetTrans
Rollback
Update

OLE Server Methods

Add
AddNew
AppendChunk
AutoBindDisable
AutoBindEnable
BeginTrans
Clone
CommitTrans
ConnectSession
CreateDynaset
CreateNamedSession
CreateSession
Delete
Edit
ExecuteSQL
FieldSize
GetChunk
LastServerErrReset
MoveFirst
MoveLast
MoveNext
MovePrevious
OpenDatabase
Refresh
Remove
ResetTrans
Rollback
Update

OLE Server Methods

Add
AddNew
AppendChunk
AutoBindDisable
AutoBindEnable
BeginTrans
Clone
CommitTrans
ConnectSession
CreateDynaset
CreateNamedSession
CreateSession
Delete
Edit
ExecuteSQL
FieldSize
GetChunk
LastServerErrReset
MoveFirst
MoveLast
MoveNext
MovePrevious
OpenDatabase
Refresh
Remove
ResetTrans
Rollback
Update

Add Method

See Also Example

Applies To

OraParameters Collection

Description

Adds a parameter to the **OraParameters** collection.

Usage

*oraparameters***.Add** Name, Value, IOType

Arguments

Name	The name of the parameter to be added to the parameters collection. This name is used both for parameter identification and as the placeholder in associated SQL and PL/SQL statements.
Value	A variant specifying the initial value of the parameter. The initial value of the parameter is significant, as it defines the data type of the parameter.
IOType	An integer code specifying how the parameter is to be used in SQL statements and PL/SQL blocks.

The **IOType** settings are:

Constant	Value	Description
ORAPARM_INPUT	1	Input variable.
ORAPARM_OUTPUT	2	Output variable.
ORAPARM_BOTH	3	Both an input and an output variable.

These values can be found in the file **ORACONST.TXT**.

Remarks

Use parameters to represent SQL bind variables (as opposed to rebuilding the SQL statement). SQL bind variables are especially useful because you can change a parameter value without having to reparse the query. Use SQL bind variables only as input variables.

You can also use parameters to represent PL/SQL bind (in/out) variables. You can use PL/SQL bind variables as both input and output variables.

See Also

OraParameter Object
OraParameters Collection
Remove Method

Add Method Example

This example demonstrates the **Add** and **Remove** parameter methods, the ServerType parameter property and the the **ExecuteSQL** database method to call a Stored Procedure and Function (located in ORAEXAMP.SQL). Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object.
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add EMPNO as an Input/Output parameter and set its initial value.
 OraDatabase.Parameters.Add "EMPNO", 7369, ORAPARM_INPUT
 OraDatabase.Parameters("EMPNO").ServerType = ORATYPE_NUMBER

 'Add ENAME as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "ENAME", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("ENAME").ServerType = ORATYPE_VARCHAR2

 'Add SAL as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "SAL", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("SAL").ServerType = ORATYPE_NUMBER

 'Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME.
 ' This Stored Procedure can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")
 'Display the employee number and name.

 'Execute the Stored Function Employee.GetSal to retrieve SAL.
 ' This Stored Function can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("declare SAL number(7,2);
Begin :SAL:=Employee.GetEmpSal (:EMPNO); end;")

 'Display the employee name, number and salary.
 MsgBox "Employee " & OraDatabase.Parameters("ENAME").value & ", #" &
OraDatabase.Parameters("EMPNO").value & ",Salary=" &
OraDatabase.Parameters("SAL").value

 'Remove the Parameters.
 OraDatabase.Parameters.Remove "EMPNO"
 OraDatabase.Parameters.Remove "ENAME"
 OraDatabase.Parameters.Remove "SAL"

End Sub
```

AddNew Method

See Also Example

Applies To

OraDynaset Object

Description

Clears the copy buffer and begins a record insertion operation into the specified dynaset and associated database.

Usage

oradynaset.**AddNew**
oradynaset.**DbAddNew** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

Upon initiation of a **AddNew** operation, values of fields present within the dynaset are maintained in a *copy buffer* and do not yet reflect the actual contents of the database.

The values of the fields are modified through the **OraField** object, and committed with **Update** or when database movement occurs, which discards the new row. Field values that have not been explicitly assigned are either set to NULL or allowed to default by way of the Oracle default mechanism, depending on the Column Defaulting mode of the options flag used when the **OpenDatabase** method was called. In either case, fields that appear in the database table but not in the dynaset are always defaulted by the Oracle default mechanism.

Internally, records are inserted by **AddNew** using the INSERT into TABLE (...) VALUES (...) SQL statement, and are therefore added to the end of the table.

Note: A call to **Edit**, **AddNew**, or **Delete**, will cancel any outstanding **Edit** or **AddNew** calls before proceeding. Any outstanding changes not saved using **Update** will be lost during the cancellation.

See Also

- **Delete** Method
- **Edit** Method
- **EditMode** Property
- Long and Long Raw Columns
- **MoveFirst, MoveLast, MoveNext, MovePrevious** Methods
- **Update** Method
- **Validate** Event

AddNew Method Example

This example demonstrates the use of **AddNew** and **Update** to add a new record to a dynaset.   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Begin an AddNew.
 OraDynaset.DbAddNew

 'Set the field(column) values.
 OraDynaset.Fields("EMPNO").Value = "1000"
 OraDynaset.Fields("ENAME").Value = "WILSON"
 OraDynaset.Fields("JOB").Value = "SALESMAN"
 OraDynaset.Fields("MGR").Value = "7698"
 OraDynaset.Fields("HIREDATE").Value = "19-SEP-92"
 OraDynaset.Fields("SAL").Value = 2000
 OraDynaset.Fields("COMM").Value = 500
 OraDynaset.Fields("DEPTNO").Value = 30

 'End the AddNew and Update the dynaset.
 OraDynaset.DbUpdate

 MsgBox "Added one new employee."

End Sub
```

AppendChunk Method

See Also Example

Applies To

OraField Object

Description

Appends data from a string to a LONG or LONG RAW field in the copy buffer.

Usage

*orafield***.AppendChunk**(*string*)
*orafield***.DbAppendChunk**(*string*) (Required for Visual Basic 3.0 and Access 2.0 users)

Arguments

string	Data to append to the specified field.

Remarks

AppendChunk allows the manipulation of data fields that are larger than 64KB.

See Also

FieldSize Method
GetChunk Method
Long and Long Raw Columns
OraField Object
Type Property

AppendChunk Method Example

This example demonstrates the use of **AppendChunk** to read a file into a LONG RAW column of a database. This example expects a valid dynaset named OraDynaset representing a table with a column named longraw. Copy this code into the definition section of a form. Call this procedure with a valid filename.

```
Sub AppendChunkExample (FName As String)

 'Declare various variables.
 Dim NumChunks As Integer, RemChunkSize As Integer
 Dim TotalSize As Long, CurChunk As String
 Dim I As Integer, FNum As Integer, ChunkSize As Integer

 'Set the size of each chunk.
 ChunkSize = 10240

 frmChunk.MousePointer = HOURGLASS

 'Begin an add operation.
 OraDynaset.DbAddNew

 'Clear the LONGRAW field.
 OraDynaset.Fields("LONGRAW").Value = ""

 'Get a free file number.
 FNum = FreeFile

 'Open the file.
 Open FName For Binary As #FNum

 'Get the total size of the file.
 TotalSize = LOF(FNum)

 'Set number of chunks.
 NumChunks = TotalSize \ ChunkSize

 'Set number of remaining bytes.
 RemChunkSize = TotalSize Mod ChunkSize

 'Loop through the file.
 For I = 0 To NumChunks

  'Calculate the new chunk size.
  If I = NumChunks Then
   ChunkSize = RemChunkSize
  End If

  CurChunk = String$(ChunkSize, 32)

  'Read a chunk from the file.
  Get #FNum, , CurChunk
  'Append chunk to LONGRAW field.
  OraDynaset.Fields("LONGRAW").DbAppendChunk (CurChunk)
 Next I

'Complete the add operation and update the database.
OraDynaset.DbUpdate

 'Close the file.
 Close FNum

 frmChunk.MousePointer = DEFAULT

End Sub
```

AutoBindDisable Method

See Also Example

Applies To

OraParameter Object

Description

Resets the AutoBind status of a parameter.

Usage

oraparameter.**AutoBindDisable**

Remarks

If a parameter has **AutoBindDisabled** status, it is not automatically bound to a SQL or PL/SQL statement.

See Also

AutoBindEnable Method
OraParameter Object

AutoBindDisable Method Example

This example demonstrates the use of **AutoBindDisable** and **AutoBindEnable** to prevent unnecessary parameter binding while creating various dynasets that use different parameters. Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As object
 Dim OraDatabase As object
 Dim OraDynaset As object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add the job input parameter with initial value MANAGER.
 OraDatabase.Parameters.Add "job", "MANAGER", 1

 'Add the deptno input parameter with initial value 10.
 OraDatabase.Parameters.Add "deptno", 10, 1

 'Disable the deptno parameter for now.
 OraDatabase.Parameters("deptno").AutoBindDisable

 'Create the OraDynaset Object using the job parameter.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where job
= :job", 0&)

 'Only employees with job=MANAGER will be contained in the dynaset.
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", Job=" &
OraDynaset.Fields("job").value


 'Enable the deptno parameter and disable the job parameter.
 OraDatabase.Parameters("deptno").AutoBindEnable
 OraDatabase.Parameters("job").AutoBindDisable

 'Create the OraDynaset Object using the deptno parameter.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where deptno
= :deptno", 0&)

 'Only employees with deptno=10 will be contained in the dynaset.
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", DeptNo=" &
OraDynaset.Fields("deptno").value

End Sub
```

AutoBindEnable Method

See Also Example

Applies To

OraParameter Object

Description

Sets the AutoBind status of a parameter.

Usage

*oraparameter***.AutoBindEnable**

Remarks

If a parameter has **AutoBindEnabled** status, it is automatically bound to a SQL or PL/SQL statement.

See Also

AutoBindDisable Method
OraParameter Object

AutoBindEnable Method Example

This example demonstrates the use of **AutoBindDisable** and **AutoBindEnable** to prevent unnecessary parameter binding while creating various dynasets that use different parameters. Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As object
 Dim OraDatabase As object
 Dim OraDynaset As object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add the job input parameter with initial value MANAGER.
 OraDatabase.Parameters.Add "job", "MANAGER", 1

 'Add the deptno input parameter with initial value 10.
 OraDatabase.Parameters.Add "deptno", 10, 1

 'Disable the deptno parameter for now.
 OraDatabase.Parameters("deptno").AutoBindDisable

 'Create the OraDynaset Object using the job parameter.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where job
= :job", 0&)

 'Only employees with job=MANAGER will be contained in the dynaset.
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", Job=" &
OraDynaset.Fields("job").value


 'Enable the deptno parameter and disable the job parameter.
 OraDatabase.Parameters("deptno").AutoBindEnable
 OraDatabase.Parameters("job").AutoBindDisable

 'Create the OraDynaset Object using the deptno parameter.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where deptno
= :deptno", 0&)

 'Only employees with deptno=10 will be contained in the dynaset.
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", DeptNo=" &
OraDynaset.Fields("deptno").value

End Sub
```

BeginTrans Method

See Also Example

Applies To

OraSession Object

Description

Begins a database transaction within the specified session.

Usage

orasession.**BeginTrans**
orasession.**DbBeginTrans** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

Once this method has been called, no database transactions are committed until a **CommitTrans** is issued. Alternatively, the session can be rolled back using **Rollback**. If a transaction has already been started, repeated use of **BeginTrans** causes an error.

See Also

CommitTrans Method
OraSession Object
ResetTrans Method
Rollback Method

BeginTrans Method Example

This example demonstrates the use of **BeginTrans** to group a set of dynaset edits into a single transaction and **Rollback** to cancel those changes.   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object
 Dim fld As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Start Transaction processing.
 OraSession.DbBeginTrans

 'Setup a field object to save object references.
 Set fld = OraDynaset.Fields("sal")

 'Traverse until EOF is reached, setting
 'each employees salary to zero.
 Do Until OraDynaset.EOF = True
   OraDynaset.DbEdit
   fld.value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'Currently, the changes have NOT been committed
 'to the database.

 'End Transaction processing.
 'Using RollbackTrans means the rollback can
 'be canceled in the Validate event.
 OraSession.DbRollback
 'MsgBox "Salary changes rolled back."

End Sub
```

Clone Method

See Also

Applies To

OraDynaset Object

Description

Returns a duplicate dynaset of the specified dynaset.

Usage

Set *oradynaset2* = *oradynaset1***.Clone**
Set *oradynaset2* = *oradynaset1***.DbClone** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

The method creates a duplicate dynaset of the one specified.   Each dynaset has its own current record.   Using **Clone** has no effect on the original dynaset.   You cannot add, update, or remove records from a dynaset clone.

Use **Clone** to perform an operation on a dynaset that requires multiple current records.

Bookmarks of a dynaset and its clone are interchangeable; bookmarks of dynasets created with separate **CreateDynaset** methods are not interchangeable.

See Also

BookMark Property
CreateDynaset Method

Close Method

See Also

Remarks

Neither the **OraDatabase** nor the **OraDynaset** object supports this method. Once an **OraDatabase** or **OraDynaset** object has gone out of scope and there are no references to it, it closes automatically.

See Also

CreateDynaset Method
OpenDatabase Method
OraDatabase Object
OraDynaset Object

CommitTrans Method

See Also Example

Applies To

OraConnection Object, **OraSession** Object

Description

Ends the current transaction and commits all pending changes to the database.

Usage

*oraconnection***.CommitTrans**
*oraconnection***.DbCommitTrans** (Required for Visual Basic 3.0 and Access 2.0 users)
*orasession***.CommitTrans**
*orasession***.DbCommitTrans** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

oraconnection.**CommitTrans** commits all pending transactions for the specified connection. If not participating in a session-wide transaction (by using the session object's **BeginTrans** method) this method has no effect, because all operations are autocommitted whenever they are performed. However, when a session-wide transaction is in progress, you can use this method to commit the transactions for the specified connection prematurely.

Use this method with care. It undermines the normal operation of session-wide transactions. In many cases you can use the **OraSession** object's transaction protocol instead.

orasession.**CommitTrans** commits all transactions present within the session. **CommitTrans** is valid only when a transaction has been started with **BeginTrans**. If a transaction has not been started, use of **CommitTrans** causes an error.

Each connection within the session is committed separately, but no two-phase commit is used. This means that if one connection fails to commit, the session may be partially committed. However, the **OraSession** object is designed to serve the purpose of a transaction manager object; in the future, two-phase commit may be implemented among all connections within the transaction.

If transaction integrity is necessary, the application developer should ensure that all databases are opened using the same Connect and DatabaseName properties and thus share only one connection.

See Also

BeginTrans Method
OraConnection Object
OraSession Object
ResetTrans Method
Rollback Method

CommitTrans Method Example

This example demonstrates the use of **BeginTrans** to group a set of dynaset edits into a single transaction and **ComitTrans** to accept the changes..   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object
 Dim fld As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Start Transaction processing.
 OraSession.DbBeginTrans

 'Setup a field object to save object references.
 Set fld = OraDynaset.Fields("sal")

 'Traverse until EOF is reached, setting
 'each employees salary to zero.
 Do Until OraDynaset.EOF = True
   OraDynaset.DbEdit
   fld.value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'Currently, the changes have NOT been committed
 'to the database.

 'End Transaction processing.
 'Commit the changes to the database
 OraSession.DbCommitTrans
 MsgBox "Salary changes committed."

End Sub
```

ConnectSession Method

See Also Example

Applies To

OraSession Object

Description

Returns the **OraSession** object with the specified name that is associated with the specified session's **OraClient** object.

Usage

Set *orasession2* = *orasession1***.ConnectSession**(*session_name*)

Arguments

session_name	A string, specifying the name of the session.

Remarks

This method is provided for simplicity and is equivalent to iterating through the **OraSessions** collection of the current session's **OraClient** object and searching for a session named *session_name*. The **OraSessions** collection will contain only sessions created via the current application and no others. This means that it is not possible to share sessions across applications, only within applications.

See Also

CreateSession Method
OraClient Object
OraSession Object
OraSessions Collection

ConnectSession Method Example

This example demonstrates the use of **ConnectSession** and **CreateNamedSession** to allow an application to use a session it previously created, but did not save.   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim dfltsess As object
 Dim OraSession As object

 'Create the default OraSession Object.
 Set dfltsess = CreateObject("OracleInProcServer.XOraSession")

 'Try to connect to "ExampleSession". If it does not exist
 'an error is generated.
 On Error GoTo SetName
 Set OraSession = dfltsess.ConnectSession("ExampleSession")
 On Error GoTo 0

 'You can specify other processing here, such as creating a
 ' database and/or dynaset.

Exit Sub

SetName:
'The session named "ExampleSession" was not found, so create it.
Set OraSession = dfltsess.Client.CreateSession("ExampleSession")
Resume Next

End Sub
```

CreateDynaset Method

See Also Example

Applies To

OraDatabase Object

Description

Creates an OraDynaset object from the specified SQL SELECT statement and options.

Usage

Set *oradynaset* = *oradatabase*.**CreateDynaset*(***sql_statement, options***)**
Set *oradynaset* = *oradatabase*.**DbCreateDynaset*(***sql_statement, options***)**
(Required for Visual Basic 3.0 and Access 2.0 users)

Arguments

sql_statement	Any valid Oracle SQL SELECT statement.
options	A bit flag indicating the status of any optional states of the dynaset. You can combine one or more options by adding their respective values.

The options flag values are:

Constant	**Value**	**Description**
ORADYN_DEFAULT	&H0&	Accept the default behavior.
ORADYN_NO_AUTOBIND	&H1&	Do not perform automatic binding of database parameters.
ORADYN_NO_BLANKSTRIP	&H2&	Do not strip trailing blanks from character string data retrieved from the database.
ORADYN_READONLY	&H4&	Force dynaset to be read-only.
ORADYN_NOCACHE	&H8&	Do not create a local dynaset data cache. Without the local cache, previous rows within a dynaset are unavailable; however, increased performance results during retrieval of data from the database (move operations) and from the rows (field operations). Use this option in applications that make single passes through the rows of a dynaset for increased performance and decreased resource usage.

These values can be found in the file **ORACONST.TXT**.

Remarks

The SQL statement must be a SELECT statement or an error is returned. Features such as views, synonyms, column aliases, schema references, table joins, nested selects and remote database references can be used freely. Object names are not modified in any way (standard Oracle rules on case apply).

The updatability of the resultant dynaset depends on the Oracle SQL rules of updatability, on the access you have been granted, and on the options flag. For the dynaset to be updatable, three conditions must be met: (1) a SQL statement must refer to a simple column list or to the entire column list (*), the statement must not set the read-only flag of the options argument, and (3) Oracle must permit ROWID references to the selected rows of the query. Any SQL statement that does not meet these criteria is processed, but the results are not updatable and the dynasets **Updatable** property returns **False**.

This method automatically moves to the first row of the created dynaset.

You can use SQL bind variables in conjunction with the **OraParameters** collection.

See Also

Clone Method
Long and Long Raw Columns
MoveFirst Method
MovePrevious Method
OpenDatabase Method
OraDatabase Object
OraDynaset Object
OraParameter Object
OraParameters Collection
Updatable Property

CreateDynaset Method Example

This example demonstrates CreateObject, **OpenDatabase** and **CreateDynaset**. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select empno, ename from emp",
0&)

 'Display the first record.
 MsgBox "Employee " & OraDynaset.Fields("empno").value & ", #" &
OraDynaset.Fields("ename").value

End Sub
```

CreateNamedSession Method

See Also Example

Applies To

OraSession Object

Description

Creates and returns a new named **OraSession** Object.

Usage

orasession = *orasession***.CreateNamedSession**(*session_name*)

Arguments

session_name	A string specifying the name of the session.

Remarks

Using this method, you can create named sessions that can be referenced later in the same application without having to explicitly save the **OraSession** object when it is created. Once a session has been created, the application can reference it by way of the **ConnectSession** method or the **OraSessions** collection of their respective **OraClient** object. The **OraSessions** collection only contains sessions created within the current application. This means that it is not possible to share sessions across applications, only within applications.

This method is provided for simplicity and is equivalent to the **CreateSession** method of the **OraClient** object.

See Also

ConnectSession Method
CreateSession Method
OraClient Object
OraSession Object
OraSessions Collection

CreateNamedSession Method Example

This example demonstrates the use of **ConnectSession** and **CreateNamedSession** to allow an application to use a session it previously created, but did not save. Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim dfltsess As object
 Dim OraSession As object
 Dim OraDatabase As Object
Dim OraDynaset As Object

 'Create the default OraSession Object.
 Set dfltsess = CreateObject("OracleInProcServer.XOraSession")

 'Try to connect to "ExampleSession". If it does not exist
 'an error is generated.
 On Error GoTo SetName
 Set OraSession = dfltsess.ConnectSession("ExampleSession")
 On Error GoTo 0

'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Display or manipulate data here

Exit Sub

SetName:
'The session named "ExampleSession" was not found, so create it.
Set OraSession = dfltsess.CreateNamedSession("ExampleSession")
Resume Next

End Sub
```

CreateSession Method

See Also   Example

Applies To

OraClient Object

Description

Creates a new named OraSession object.

Usage

orasession = *oraclient***.CreateSession*(****session_name****)***

Arguments

session_name	A string specifying the name of the session.

Remarks

Using this method, you can create named sessions that can be referenced later in the same application without having to explicitly save the **OraSession** object when it is created.   Once a session has been created, the application can reference it by way of the **ConnectSession** method or the **OraSessions** collection of their respective **OraClient** object.   The **OraSessions** collection only contains sessions created within the current application.   This means that it is not possible to share sessions across applications, only within applications.

See Also

OraClient Object
OraSession Object

CreateSession Method Example

This example demonstrates how to create an session object using the CreateSession method of client object.   Copy and paste this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraClient As Object
 Dim OraSession As Object
 Dim NamedOraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Get the OraClient object.
 Set OraClient = OraSession.Client

 'Create a named OraSession Object
 'Alternatively, you could use the CreateNamedSession
 'method of the OraSession Object.
 Set NamedOraSession = OraClient.CreateSession("ExampleSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = NamedOraSession.OpenDatabase("ExampleDb", "scott/tiger",
0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

End Sub
```

Delete Method

See Also Example

Applies To

OraDynaset Object

Description

Deletes the current row of the specified dynaset.

Usage

*oradynaset***.Delete**
*oradynaset***.DbDelete** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

A row must be current before you can use **Delete**; otherwise an error occurs.

Any references to the deleted row produce an error. The deleted row, as well as the next and previous rows, remain current until database movement occurs (using the **MoveFirst**, **MovePrevious**, **MoveNext**, or **MoveLast** methods). Once such movement occurs, you cannot make the deleted row current again.

You cannot restore deleted records except by using transactions.

Note: A call to **Edit**, **AddNew**, or **Delete**, will cancel any outstanding **Edit** or **AddNew** calls before proceeding. Any outstanding changes not saved using **Update** will be lost during the cancellation.

See Also

AddNew Method
BeginTrans Method
CommitTrans Method
Edit Method
MoveFirst, MoveLast, MoveNext, MovePrevious Methods
ResetTrans Method
Rollback Method

Delete Method Example

This example demonstrates the use of **Delete** to remove records from a database. Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("exampledb", "scott/tiger", 0&)

'Create the OraDynaset Object.
'Only select the employees in Department 10.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where
deptno=10", 0&)

 Do Until OraDynaset.EOF
   OraDynaset.DbDelete
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All employees from department 10 removed."

End Sub
```

Edit Method

See Also Example

Applies To

OraDynaset Object

Description

Begins an edit operation on the current row by copying the data to the copy buffer.

Usage

oradynaset.**Edit**
oradynaset.**DbEdit** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

Edit causes the locally cached data to be compared with the corresponding row of the Oracle database. An error will be generated if the Oracle database data is not the same as the data currently being browsed. If this operation succeeds, the row is locked using SELECT ... FOR UPDATE until the edit is completed with **Update** or until database movement occurs, which discards any edits in progress. The behavior of the SELECT ... FOR UPDATE is affected by the Lock Wait mode of the options flag used when the **OpenDatabase** method was called.

During editing, changes made to fields are kept in a shadowed copy buffer and do not yet reflect the actual contents of the database. However, all references to the row return the newly modified data as long as the edit operation is still in progress.

When data is modified within a data control attached to this dynaset, the **Edit** method is invoked automatically upon the next record movement. Thus, this method is required only when modifications are made to field data within code.

Note: A call to **Edit**, **AddNew**, or **Delete**, will cancel any outstanding **Edit** or **AddNew** calls before proceeding. Any outstanding changes not saved using **Update** will be lost during the cancellation.

See Also

AddNew Method
CreateDynaset Method
Delete Method
OpenDatabase Method
Locks and Editing
Long and Long Raw Columns
Update Method
Validate Event.

Edit Method Example

This example demonstrates the use of **Edit** and **Update** to update values in a database. Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)
'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Traverse until EOF is reached, setting
 'each employees salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

End Sub
```

ExecuteSQL Method

See Also Example

Applies To

OraDatabase Object

Description

Executes a single non-SELECT SQL statement or a PL/SQL block.

Usage

rowcount = *oradatabase*.**ExecuteSQL**(*sql_statement*)
rowcount = *oradatabase*.**DbExecuteSQL**(*sql_statement*) (Required for Visual Basic 3.0 and Access 2.0 users)

Arguments

sql_statement	Any valid Oracle non-SELECT SQL statement.

Remarks

Executes a SQL statement and returns the number of rows processed by that statement.

The *sql_statement* can be one continuous line with no breaks. If it is necessary to break the line, be sure to use line feeds (ASCII 10). Do not use carriage returns (ASCII 13), because the underlying Oracle database functions treat carriage returns as null terminators.

You can use PL/SQL bind variables in conjunction with the **OraParameters** collection.

When executing PL/SQL blocks or calling stored procedures, you must include a BEGIN and END around your call as if you were executing an anonymous PL/SQL block. This is equivalent to the EXECUTE command of SQL*Plus and SQL*DBA.

Note: **ExecuteSQL** should be used with care since any SQL statement or PL/SQL block that is executed can adversely affect currently open dynasets. This is especially true if the **OraDatabase** object used for the **ExecuteSQL** method is the same as the one that was used to create the dynaset. Use a different **OraDatabase** object if you are unsure.

Normal dynaset operations can be adversely affected, if in transactional mode, a database commit is issued. This can happen if either a SQL commit command or a Data Control Language (DCL) or Data Definition Language (DDL) command is issued. DCL and DDL SQL commands, such as CREATE, DROP, ALTER, GRANT and REVOKE always force a commit, which in turn commits everything done before them. Consult the Oracle7 Server SQL Language Reference Manual for more details about DCL, DDL and transactions.

Data Type

Long Integer

See Also

CreateDynaset Method
OraDatabase Object
OraParameters Collection
Transactions

ExecuteSQL Method Example

This example demonstrates the **Add** and **Remove** parameter methods, the ServerType parameter property and the the **ExecuteSQL** database method to call a Stored Procedure and Function (located in ORAEXAMP.SQL).   Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object.
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add EMPNO as an Input/Output parameter and set its initial value.
 OraDatabase.Parameters.Add "EMPNO", 7369, ORAPARM_INPUT
 OraDatabase.Parameters("EMPNO").ServerType = ORATYPE_NUMBER

 'Add ENAME as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "ENAME", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("ENAME").ServerType = ORATYPE_VARCHAR2

 'Add SAL as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "SAL", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("SAL").ServerType = ORATYPE_NUMBER

 'Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME.
 ' This Stored Procedure can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")
 'Display the employee number and name.

 'Execute the Stored Function Employee.GetSal to retrieve SAL.
 ' This Stored Function can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("declare SAL number(7,2);
Begin :SAL:=Employee.GetEmpSal (:EMPNO); end;")

 'Display the employee name, number and salary.
 MsgBox "Employee " & OraDatabase.Parameters("ENAME").value & ", #" &
OraDatabase.Parameters("EMPNO").value & ",Salary=" &
OraDatabase.Parameters("SAL").value

 'Remove the Parameters.
 OraDatabase.Parameters.Remove "EMPNO"
 OraDatabase.Parameters.Remove "ENAME"
 OraDatabase.Parameters.Remove "SAL"

End Sub
```

FieldSize Method

See Also

Applies To

OraField Object

Description

Returns the number of bytes stored in a LONG or LONG RAW field.   Not available at design time and read-only at run time.

Usage

data_size = *orafield***.FieldSize( )**
data_size = *orafield***.DbFieldSize( )** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

Returns the number of bytes stored in a LONG or LONG RAW field, up to a value of around 64KB.   If the field contains more than 64KB, **FieldSize** returns -1 (negative one).

Oracle does not return the length of columns that are greater than 64KB, so the only way to determine the length is to retrieve the column.   To conserve resources, columns of length greater than 64KB are not retrieved automatically.

Data Type

Long Integer

See Also

AppendChunk Method
GetChunk Method
Long and Long Raw Columns
OraField Object
Type Property

FindFirst, FindLast, FindNext, FindPrevious Methods

See Also

Remarks

The **OraDynaset** object does not support these methods.

See Also

MoveFirst, MoveLast, MoveNext, MovePrevious Methods
OraDatabase Object

GetChunk Method

See Also Example

Applies To

OraField Object

Description

Returns a string containing the bytes of all or a portion of a long or long raw field.

Usage

data_string = *orafield***.GetChunk**(*offset, numbytes*)
data_string = *orafield***.DbGetChunk**(*offset, numbytes*) (Required for Visual Basic 3.0 and Access 2.0 users)

Arguments

offset	The number of bytes of the field to skip before copying data.
numbytes	The number of bytes to copy.

Remarks

When possible, **GetChunk** retrieves the specified bytes from the local cache. However, to conserve resources, some of the data may not be stored locally. In these cases, **GetChunk** requests the necessary data from the database as required. As part of this process, data from all fields (except the Long or Long Raw field) in the dynaset are retrieved and compared with the cached values for consistency. If any changes have occurred since the fetch of the original partial data, then **GetChunk** aborts the operation with an error. In the case of an abort, the returned string is NULL.

If a long or long raw field is less than 64KB in size, it is quicker to retrieve the data using the **Value** property than using **GetChunk**.

See Also

AppendChunk Method
FieldSize Method
Long and Long Raw Columns
OraField Object
Type Property
Value Property

GetChunk Method Example

This example demonstrates the use of **GetChunk** to retrieve a LONG RAW column of a database and save it into a file. This example expects a valid dynaset named OraDynaset representing a table with a column named 'longraw'. Copy and paste this code into the definition section of a form. Call this procedure with a valid filename.

```
Sub GetChunkExample (FName As String)

'Declare various variables
Dim CurSize As Integer, ChunkSize  As Long
Dim I As Integer, FNum As Integer, CurChunk As String

'Set the size of each chunk
ChunkSize = 10240

frmChunk.MousePointer = HOURGLASS

'Get a free file number
FNum = FreeFile

'Open the file
Open FName For Binary As #FNum

 I = 0
'Loop through all of the chunks
'Oracle does not return the size of columns > 64KB.
'We should loop until the length of our block is
'less than we asked for.
Do
 CurChunk = OraDynaset.Fields("LONGRAW").DbGetChunk(I * ChunkSize, ChunkSize)
 CurSize = Len(CurChunk) 'Get the length of the current chunk.
 Put #FNum, , CurChunk   'Write chunk to file.
 I = I + 1
Loop Until CurSize < ChunkSize

'Close the file.
Close FNum

frmChunk.MousePointer = DEFAULT

End Sub
```

LastServerErrReset Method

See Also

Applies To

OraDatabase Object, **OraSession** Object

Description

Clears the **LastServerErr** to a zero value and sets **LastServerErrText** to NULL for the specified object.

Usage

oradatabase.**LastServerErrReset**
orasession.**LastServerErrReset**

Remarks

This method allows user programs to better determine which program requests generated the Oracle error.

See Also

LastServerErr Property
LastServerErrText Property
OraDatabase Object
OraSession Object

MoveFirst, MoveLast, MoveNext, MovePrevious Methods

See Also Example

Applies To

OraDynaset Object

Description

These methods change the cursor position to the first, last, next, or previous row within the specified dynaset.

Usage

oradynaset.**MoveFirst**
oradynaset.**DbMoveFirst** (Required for Visual Basic 3.0 and Access 2.0 users)
oradynaset.**MoveLast**
oradynaset.**DbMoveLast** (Required for Visual Basic 3.0 and Access 2.0 users)
oradynaset.**MovePrevious**
oradynaset.**DbMovePrevious** (Required for Visual Basic 3.0 and Access 2.0 users)
oradynaset.**MoveNext**
oradynaset.**DbMoveNext** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

The data control buttons map (from left to right or from top to bottom) to the **MoveFirst**, **MovePrevious**, **MoveNext**, and **MoveLast** methods. The **BOF** and **EOF** properties are never **True** when using the data control buttons.

When the first or last record is current, record movement does not occur if you use **MoveFirst** or **MoveLast**, respectively.

You force the query to completion if you use **MoveLast** on a dynaset.

If you use **MovePrevious** and the first record is current, there is no current record and **BOF** is **True**. Using **MovePrevious** again causes an error, though **BOF** remains **True**.

If you use **MoveNext** and the last record is current, there is no current record and **EOF** is **True**. Using **MoveNext** again causes an error, though **EOF** remains **True**.

When you open a dynaset, **BOF** is **False** and the first record is current. If a dynaset is empty, **BOF** and **EOF** are both **True** and there is no current record.

If an **Edit** or **AddNew** operation is pending and you use one of the **Move** methods indirectly by way of the data control, then **Update** is invoked automatically (although it may be stopped during the **Validate** event).

If an **Edit** or **AddNew** operation is pending and you use one of the **Move** methods directly without the data control, pending **Edit** or **AddNew** operations cause existing changes to be lost, though no error occurs.

Data is fetched from the database as necessary, so performing a **MoveFirst** followed by **MoveNext** will incrementally build the mirrored (cached) local set without requiring read-ahead of additional data. However, executing a **MoveLast** will require that the entire query be evaluated and stored locally.

When a dynaset is attached to a data control, these methods first notify the data control's **Validate** event that record motion is about to occur.   The **Validate** handler may deny the request for motion, in which case the request is ignored.   If the record pointer is successfully moved, then all custom controls attached to the data control are notified automatically of the new record position.

See Also

AddNew Method
BOF Property
Edit Method
EditMode Property
EOF Property
RecordCount Property
Tuning and Customization
Update Method
Validate Event

MoveFirst, MoveLast, MoveNext, MovePrevious Methods Example

This example demonstrates record movement within a dynaset using **MoveFirst**, **MoveNext**, **MoveLast**, **MovePrevious**. Copy and paste this code into the definition section of a form.. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select empno, ename from emp",
0&)

MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", " &
OraDynaset.Fields("ename").value

 'Move to the next record and display it.
 OraDynaset.DbMoveNext
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", " &
OraDynaset.Fields("ename").value

 'Move to the last record and display it.
 OraDynaset.DbMoveLast
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", " &
OraDynaset.Fields("ename").value

 'Move to the previous record and display it.
 OraDynaset.DbMovePrevious
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", " &
OraDynaset.Fields("ename").value


End Sub
```

OpenDatabase Method

See Also Example

Applies To

OraSession Object

Description

Creates an **OraDatabase** object using the given database name and connect string and reflecting the specified options.

Usage

Set *oradatabase* = *orasession*.**OpenDatabase(***database_name, connect_string, options***)**
Set *oradatabase* = *orasession*.**DbOpenDatabase(***database_name, connect_string, options***)** (Required for Access 2.0 users)

Arguments

database_name	The Oracle SQL*Net specifier used when connecting the data control to a database.
connect_string	The username and password to be used when connecting to an Oracle database.
options	A bit flag word used to set the optional modes of the database. If *options* = 0, the default mode settings will apply. The following modes are available:

Column Defaulting mode.
The default mode is called VB mode. In VB mode, field (column) values not explicitly set are set to NULL when using **AddNew** or **Edit**.
Optionally you can use Oracle mode. Oracle mode indicates that changes made to fields (columns) are immediately reflected in the local mirror by retrieving the changed row from the database, thus allowing Oracle to set defaults for the columns and perform required calculations.
Column Defaulting mode affects the behavior of the **AddNew** and **Edit** methods.

Lock Wait mode.
The default mode is called Wait mode. In Wait mode, when dynaset rows are about to be modified (using **Edit**), the existing row in the database is retrieved using SELECT ... FOR UPDATE to lock the row in the database. If the row about to be changed has been locked by another process (or user), the SELECT ... FOR UPDATE, waits until the row is unlocked before proceeding.
Optionally you may use NoWait mode. NoWait mode results in an immediate return of an error code, indicating that the row about to be updated is locked.
Lock Wait mode also affects any SQL statements processed using **ExecuteSQL**.

Examples of valid *database_name* arguments include:

"t:oracle:PROD"
"p:oracle7:demo"
"x:orasrv"
"mydbalias"   (Where mydbalias represents t:mfg:prod)

Examples of valid *connect_string* arguments include:

"scott/tiger"
"system/manager"

The options flag values are:

Constant	**Value**	**Description**
ORADB_DEFAULT	&H0&	Accept the default behavior.
ORADB_ORAMODE	&H1&	Let Oracle set default field (column) values.
ORADB_NOWAIT	&H2&	Do not wait on row locks when executing a "SELECT ... FOR UPDATE".

Options may be combined by adding their respective values.

These values can be found in the file **ORACONST.TXT**.

Remarks

If another connection exists within the same **OraSession** object with the same *connect_string* and *database_name* arguments, it is automatically shared, although each database object is distinct.   An **OraConnection** object is created automatically and appears within the **OraConnections** collection of the session. Opening a database has the effect of opening a connection (if it does not already exist), but does not perform any SQL actions.

See Also

AddNew Method
Edit Method
ExecuteSQL Method
Locks and Editing
OraConnection Object
OraConnections Collection
OraDatabase Object
OraField Object
OraSession Object

OpenDatabase Method Example

This example demonstrates how to create a dynaset and all of the underlying objects programmatically. Copy and paste this code into the definition section of a form with textboxes named txtEmpNo and txtEName. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select empno, ename from emp",
0&)

 'Display the first record.
 txtEmpNo = OraDynaset.Fields("empno").value
 txtEName = OraDynaset.Fields("ename").value

End Sub
```

Refresh Method

See Also Example

Applies To

OraDynaset Object

Description

Forces an immediate update of the dynaset given the current **Connect**, **DatabaseName,** and **SQL** properties.

Usage

oradynaset.**Refresh**
oradynaset.**DbRefresh** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

This method cancels all edit operations (**Edit** and **AddNew**), executes the current contents of the SQL statement buffer, and moves to the first row of the resulting dynaset. Any dynaset objects created before issuing the **Refresh** method, including bookmarks, record counts, and field collections, are considered invalid. The **OraConnection** and **OraSession** objects associated with the previous dynaset remain unchanged.

Performing a refresh operation with this method can be more efficient than refresh operations using a data control, and provides the added capability of executing a modified SQL statement without creating a new dynaset.

Using *oradynaset*.**Refresh** is the preferred refresh method when changing parameter values, because required database operations are minimized (SQL parsing, binding, etc.). This can lead to much improved performance when only parameter values have changed.

If the SQL statement associated with the dynaset has been changed and a following **Refresh** is issued, it is possible that the specified SQL statement may not be valid. In this case, the dynaset remains a valid object but does not permit any row or field operations. In this state, bound controls exhibit unusual behaviors similar to those that occur when the standard Visual Basic data control **RecordSource** is set to an invalid SQL statement at run time and then Refreshed. A NULL or empty SQL statement is considered invalid by **Refresh**.

See Also

AddNew Method
Connect Property
CreateDynaset Method
DatabaseName Property
Edit Method
OraConnection Object
OraDynaset Object
OraSession Object
RecordSource Property
SQL Property

Refresh Method Example

This example demonstrates the use of parameters, the **Refresh** method, and the **SQL** property to restrict selected records. Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

 'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create a parameter with an initial value.
 OraDatabase.Parameters.Add "job", "MANAGER", 1

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp where
job=:job", 0&)

 'Notice that the SQL statement is NOT modified.
 MsgBox OraDynaset.SQL

 'Currently, OraDynaset only contains employees whose
 'job is MANAGER.

 'Change the value of the job parameter.
 OraDatabase.Parameters("job").Value = "SALESMAN"

 'Refresh the dynaset.
 OraDynaset.DbRefresh

 'Currently, OraDynaset only contains employees whose
 'job is SALESMAN.

 'Notice that the SQL statement is NOT modified.
 MsgBox OraDynaset.SQL

 'Remove the parameter.
 OraDatabase.Parameters.Remove ("job")

 End Sub
```

Remove Method

See Also Example

Applies To

OraParameters Collection

Description

Removes a parameter from the **OraParameters** collection.

Usage

oraparameters.**Remove**(*member_name)*

Arguments

member_name	A variant specifying an integer subscript from 0 to Count - 1, or the parameter name.

Remarks

Instead of repeatedly removing and adding unwanted parameters, use **AutoBindDisable** and **AutoBindEnable**.

See Also

Add Method
AutoBindEnable Method
AutoBindDisable Method
OraDatabase Object
OraParameter Object
OraParameters Collection

Remove Method Example

This example demonstrates the **Add** and **Remove** parameter methods, the ServerType parameter property and the the **ExecuteSQL** database method to call a Stored Procedure and Function (located in ORAEXAMP.SQL).   Copy and paste this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
 Dim OraSession As Object
 Dim OraDatabase As Object
 Dim OraDynaset As Object

 'Create the OraSession Object.
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

 'Create the OraDatabase Object.
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Add EMPNO as an Input/Output parameter and set its initial value.
 OraDatabase.Parameters.Add "EMPNO", 7369, ORAPARM_INPUT
 OraDatabase.Parameters("EMPNO").ServerType = ORATYPE_NUMBER

 'Add ENAME as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "ENAME", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("ENAME").ServerType = ORATYPE_VARCHAR2

 'Add SAL as an Output parameter and set its initial value.
 OraDatabase.Parameters.Add "SAL", 0, ORAPARM_OUTPUT
 OraDatabase.Parameters("SAL").ServerType = ORATYPE_NUMBER

 'Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME.
 ' This Stored Procedure can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("Begin Employee.GetEmpName (:EMPNO, :ENAME); end;")
 'Display the employee number and name.

 'Execute the Stored Function Employee.GetSal to retrieve SAL.
 ' This Stored Function can be found in the file ORAEXAMP.SQL.
 OraDatabase.DbExecuteSQL ("declare SAL number(7,2);
Begin :SAL:=Employee.GetEmpSal (:EMPNO); end;")

 'Display the employee name, number and salary.
 MsgBox "Employee " & OraDatabase.Parameters("ENAME").value & ", #" &
OraDatabase.Parameters("EMPNO").value & ",Salary=" &
OraDatabase.Parameters("SAL").value

 'Remove the Parameters.
 OraDatabase.Parameters.Remove "EMPNO"
 OraDatabase.Parameters.Remove "ENAME"
 OraDatabase.Parameters.Remove "SAL"

End Sub
```

ResetTrans Method

See Also Example

Applies To

OraSession Object

Description

Unconditionally rolls back all transactions and clears the transaction mode initiated by **BeginTrans**.

Usage

*orasession***.ResetTrans**

Remarks

This method does not generate events or produce errors. Since **ResetTrans** does not generate events, you cannot cancel **ResetTrans** in a **Validate** event, as you can **RollBack** or **Commit**.

See Also

BeginTrans Method
CommitTrans Method
OraSession Object
Rollback Method
Validate Event

ResetTrans Method Example

This example demonstrates the use of **BeginTrans** and **ResetTrans** to group a set of dynaset edits into a single transaction.   Copy this code into the definition section of a form. Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)
'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Start Transaction processing.
 OraDynaset.Session.DbBeginTrans

 'Traverse until EOF is reached, setting
 'each employees salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'Currently, the changes have NOT been committed
 'to the database.

 'End Transaction processing.
 'Using ResetTrans means the rollback cannot
 'be canceled in the Validate event.
 OraDynaset.Session.DbResetTrans
 MsgBox "Salary changes rolled back."

End Sub
```

Rollback Method

See Also Example

Applies To

OraConnection Object, **OraSession** Object

Description

Ends the current transaction and rolls back all pending changes to the database.

Usage

*oraconnection***.Rollback**
*oraconnection***.DbRollback** (Required for Visual Basic 3.0 and Access 2.0 users)
*orasession***.Rollback**
*orasession***.DbRollback** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

When this method is invoked, all **OraDynaset** objects that share the specified session or connection are given the opportunity to cancel the rollback request. If they do not cancel the request, they are then advised when the rollback succeeds.

This feature is useful primarily for dynasets that are created as part of an Oracle Data Control's operation. For such dynasets, the **Validate** event is sent to allow them to cancel the rollback request.

*OraConnection.***Rollback** rolls back all pending transactions within the specified connection. If not participating in a session-wide transaction (by using the session object's **BeginTrans** method), this method has no effect, because all operations are autocommitted whenever they are performed. However, when a session-wide transaction is in progress, you can use this call to prematurely roll back the transactions for the specified connection.

Use this method with care. It undermines the normal operation of session-wide transactions. In many cases, you can use the **OraSession** object's transaction protocol instead.

OraSession.**Rollback** rolls back all pending transactions within the specified session. **Rollback** is valid only when a transaction has been started using **BeginTrans**. Using **Rollback** without **BeginTrans** results in an error.

Transactions are rolled back by rolling back each connection separately. If an error occurs, some connections are rolled back and others are not. If the error can be corrected, executing **Rollback** again rolls back the remaining connections.

See Also

- **BeginTrans** Method
- **CommitTrans** Method
- **OraConnection** Object
- **OraSession** Object
- **ResetTrans** Method
- **Validate** Event

Rollback Method Example

This example demonstrates the use of **BeginTrans** and **Rollback** to group a set of dynaset edits into a single transaction.   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)
'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

 'Start Transaction processing.
 OraDynaset.Session.DbBeginTrans

 'Traverse until EOF is reached, setting
 'each employees salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.DbEdit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.DbUpdate
   OraDynaset.DbMoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'Currently, the changes have NOT been committed
 'to the database.

 'End Transaction processing.
 OraDynaset.Session.DbRollback
 MsgBox "Salary changes rolled back."

End Sub
```

Update Method

See Also Example

Applies To

OraDynaset Object

Description

Saves the copy buffer to the specified dynaset.

Usage

*oradynaset***.Update**
*oradynaset***.DbUpdate** (Required for Visual Basic 3.0 and Access 2.0 users)

Remarks

Update completes an **AddNew** or **Edit** operation and immediately commits changes to the database unless **BeginTrans** is pending for the session.

The mirrored data image is also updated so that the query does not have to be reevaluated to continue browsing and updating data.   The method used for updating the mirror image is subject to the options flag that was passed to the **OpenDatabase** that created this dynasets **OraDatabase** object.

If this dynaset is attached to a data control, then the data control's **Validate** event code may optionally cancel the update request.   If the update completes, then all bound controls associated with the dynaset are notified of the update so they can reflect the data changes automatically.

See Also

AddNew Method
Edit Method
Long and Long Raw Columns
OpenDatabase Method
OraDatabase Object
Validate Event

Update Method Example

This example demonstrates the use of **AddNew** and **Update** to add a new record to a dynaset.   Copy this code into the definition section of a form.   Then press F5.

```
Sub Form_Load ()

'Declare variables as OLE Objects.
Dim OraSession As Object
Dim OraDatabase As Object
Dim OraDynaset As Object

'Create the OraSession Object.
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.DbOpenDatabase("ExampleDb", "scott/tiger", 0&)
'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.DbCreateDynaset("select * from emp", 0&)

'Begin an AddNew.
OraDynaset.DbAddNew

'Set the field(column) values.
OraDynaset.Fields("EMPNO").Value = "1000"
OraDynaset.Fields("ENAME").Value = "WILSON"
OraDynaset.Fields("JOB").Value = "SALESMAN"
OraDynaset.Fields("MGR").Value = "7698"
OraDynaset.Fields("HIREDATE").Value = "19-SEP-92"
OraDynaset.Fields("SAL").Value = 2000
OraDynaset.Fields("COMM").Value = 500
OraDynaset.Fields("DEPTNO").Value = 30

'End the AddNew and Update the dynaset.
OraDynaset.DbUpdate

End Sub
```

Data Control

See Also   Properties   Methods   Events

Description

A data control provides a programmatic representation of a database and a graphical representation of the Move methods of the underlying dynaset (**MoveFirst**, **MovePrevious**, **MoveNext**, and **MoveLast)**.   You can bind other controls to a data control.

Remarks

A data control allows you to bind to it other controls that display a field, a record, or multiple records of the underlying dynaset.   When record movement occurs, data in bound controls stays in sync with the current record of the dynaset.

If a user changes that data in a control that is bound to a data control, the changes are automatically reflected in the underlying dynaset and database.

A data control allows you to perform most data access operations without writing any code at all. To create a dynaset with a data control, set the **Connect**, **DatabaseName**, and **RecordSource** properties and execute **Refresh**.

See Also

Connect Property
DatabaseName Property
MoveFirst, MoveLast, MoveNext, MovePrevious Methods
OraDynaset Object
RecordSource Property
Refresh Method

Properties

- AllowMoveLast
- AutoBinding
- BackColor
- Caption
- Connect
- Database
- DatabaseName
- DragIcon
- DragMode
- EditMode
- Enabled
- Exclusive
- FontBold
- FontItalics
- FontName
- FontSize
- FontStrikethru
- FontUnderline
- ForeColor
- HelpContextID
- Height
- Index
- Left
- MousePointer
- Name
- Options
- ReadOnly
- Recordset
- RecordSource
- Session
- Tag
- TrailingBlanks
- Top
- Visible
- Width

Methods

- Drag
- Move
- Refresh
- UpdateControls
- UpdateRecord
- ZOrder

Events

DragDrop	MouseMove
DragOver	MouseUp
Error	Reposition
MouseDown	Validate

Data Control Properties

- AllowMoveLast
- AutoBinding
- BackColor
- Caption
- Connect
- Database
- DatabaseName
- DragIcon
- DragMode
- EditMode
- Enabled
- Exclusive
- FontBold
- FontItalics
- FontName
- FontSize
- FontStrikethru
- FontUnderline
- ForeColor
- HelpContextID
- Height
- Index
- Left
- MousePointer
- Name
- Options
- ReadOnly
- Recordset
- RecordSource
- Session
- Tag
- TrailingBlanks
- Top
- Visible
- Width

Data Control Properties

- AllowMoveLast
- AutoBinding
- BackColor
- Caption
- Connect
- Database
- DatabaseName
- DragIcon
- DragMode
- EditMode
- Enabled
- Exclusive
- FontBold
- FontItalics
- FontName
- FontSize
- FontStrikethru
- FontUnderline
- ForeColor
- HelpContextID
- Height
- Index
- Left
- MousePointer
- Name
- Options
- ReadOnly
- Recordset
- RecordSource
- Session
- Tag
- TrailingBlanks
- Top
- Visible
- Width

Data Control Methods

- Drag
- Move
- Refresh
- UpdateControls
- UpdateRecord
- ZOrder

Data Control Methods

Drag
Move
Refresh
UpdateControls
UpdateRecord
ZOrder

Data Control Events

DragDrop	MouseMove
DragOver	MouseUp
Error	Reposition
MouseDown	Validate

Data Control Events

DragDrop	MouseMove
DragOver	MouseUp
Error	Reposition
MouseDown	Validate

AllowMoveLast Property

See Also

Applies To

Oracle Data Control

Description

Determines whether the user can move to the last record using the Data Control's **MoveLast** button.   Read/write at design time and run time.

Usage

oradata1**.AllowMoveLast** = [**True | False**]

Remarks

By default, **AllowMoveLast** is **True**, in which case the user has no restriction upon record motion, even when moving to the last record may be very time consuming.

When **AllowMoveLast** is **False**, the Data Controls **MoveLast** button is grayed out and disabled. However, once the last record has been encountered (either because the user has navigated to the end of the set, or because code has positioned the record pointer to the last record), the button is enabled. This gives the user visual feedback about whether or not the entire query has been fetched.   Setting this property to **False** does not prevent you from using the **MoveLast** method.

Changing this property has no effect until a **Refresh** method is sent to the data control.

Data Type

Integer (Boolean)

See Also

MoveLast Method
Refresh Method

AutoBinding Property

See Also Example

Applies To

Oracle Data Control.

Description

Determines whether the automatic binding of database object parameters will occur. Read/write at design time and run time.

Usage

oradata1**.AutoBinding** = [**True** | **False**]

Remarks

By default, **AutoBinding** is **True**, in which case the parameters in the **OraParameters** collection are bound to the SQL statement of the **RecordSource** property before data control refresh (before the SQL statement is executed). Technically speaking, the parameters are rebound when the recordset is re-created.

Setting **Autobinding** to **False** takes effect only if the SQL statement of the **RecordSource** property needs to be rebound and reexecuted. This is not the case when you simply change a parameter value and refresh the data control or simply refresh the recordset (the SQL statement only needs to be reexecuted). This is the case if you alter the **RecordSource** property and change the SQL statement.

Use this property to disable all parameter binding when executing a SQL statement that does not contain any parameters (using **CreateDynaset**, **Refresh** or **ExecuteSQL**).

Changing this property does not take effect until a **Refresh** method is sent to the data control (and the appropriate conditions apply). Changing this property has no effect when a *recordset*.**Refresh** is executed.

Data Type

Integer (Boolean)

See Also

Add Method
AutoBindDisable Method
AutoBindEnable Method
CreateDynaset Method
ExecuteSQL Method
OraParameter Object
OraParameters Collection
RecordSource Property
Refresh Method

AutoBinding Property Example

This example demonstrates the use of **AutoBinding** to show how it affects data control and recordset refresh.   Copy this code into the definition section of a new form, then press F5 to run.

```
Sub Form_Load ()

 'Set the username and password.
 oradata1.Connect = "scott/tiger"

 'Set the databasename.
 oradata1.DatabaseName = "ExampleDb"

 'Refresh the data control without setting the
 'recordsource. This has the effect of creating
 'the underlying database object so that parameters
 'can be added.
 oradata1.Refresh

 'Set the recordsource and use a SQL parameter for job.
 oradata1.RecordSource = "select * from emp where job = :job"

 'Add the job input parameter with initial value MANAGER.
 oradata1.Database.Parameters.Add "job", "MANAGER", 1

 'Add the deptno input parameter with initial value 10.
 oradata1.Database.Parameters.Add "deptno", 10, 1

 'Refresh the data control.
 oradata1.Refresh

 MsgBox "Employee #" & oradata1.Recordset.fields("empno") & ", Job=" &
oradata1.Recordset.fields("job")

 'Only employees with job=MANAGER will be contained
 'in the dynaset.

'Turn off Automatic parameter binding.
 oradata1.AutoBinding = False

 'Change the value of the job parameter to SALESMAN.
 oradata1.Database.Parameters("job").Value = "SALESMAN"

 'Refresh ONLY the recordset.
 oradata1.Recordset.DbRefresh

 MsgBox "Employee #" & oradata1.Recordset.fields("empno") & ", Job=" &
oradata1.Recordset.fields("job")

 'The query will still execute even with AutoBinding=False
 'because the dynaset has not been re-created.
 'Set the recordsource and use a SQL parameter for deptno.
 oradata1.RecordSource = "select * from emp where deptno = :deptno"

 On Error GoTo paramerr
 'Attempt to refresh the data control.
 'An error should occur, because AutoBind=False, the SQL
 'statement contains a parameter, and the SQL statement
 'needs to be bound before execution.
 oradata1.Refresh

Exit Sub

paramerr:
 MsgBox oradata1.Database.Session.LastServerErrText
Exit Sub

End Sub
```

BackColor Property

See Also

Applies To

Oracle Data Control.

Description

Determines the background color of an object.

See Visual Basic Help for more information.

See Also

ForeColor Property

Caption Property

Applies To

Oracle Data Control.

Description

Determines the text displayed in or next to a control.

See Visual Basic Help for more information.

Connect Property

See Also

Applies To

Oracle Data Control.

Description

The username and password to be used when connecting the data control to an Oracle database.   Read/write at design time and run time.

Usage

oradata1**.Connect** = [*username/password*]

Remarks

This string is passed to the **OpenDatabase** method of the **OraSession** object when the control is refreshed.   Changing this property does not take effect until a **Refresh** method is sent to the data control.

If the data control is refreshed and the **Connect** property has not been specified, the refresh will fail.

Examples of valid **Connect** properties include:

"scott/tiger"
"system/manager"

Data Type

String

See Also

OpenDatabase Method
OraSession Object
Refresh Method

Database Property

See Also

Applies To

Oracle Data Control.

Description

Returns the **OraDatabase** object associated with the data control.   Not available at design time and read-only at run time.

Usage

oradatabase = oradata1**.Database**

Remarks

If the data control has not been refreshed, any references to this property results in an "Object variable not set" runtime error.

Changing this property has no effect until a **Refresh** method is sent to the data control.

Data Type

OLE Object (**OraDatabase**)

See Also

OraDatabase Object
Refresh Method

DatabaseName Property

See Also

Applies To

Oracle Data Control.

Description

The Oracle SQL*Net specifier used when connecting the data control to an Oracle database.   Read/write at design time and run time.

Usage

oradata1**.DatabaseName** = [DatabaseName]

Remarks

The Oracle SQL*Net specifier should include the Oracle SQL*Net protocol identifier, Oracle database name, and optional database instance (SQL*Net aliases can also be used).   This string is passed to the **OpenDatabase** method of the **OraSession** object when the control is refreshed.   Changing this property does not take effect until a **Refresh** method is sent to the data control.

If the data control is refreshed and **DatabaseName** has not been specified, the refresh fails.

Examples of valid **DatabaseName** properties include:

"t:oracle:PROD"
"p:oracle7:demo"
"x:orasrv"
"mydbalias"   (Where mydbalias represents "t:mfg:prod")

Data Type

String

See Also

OpenDatabase Method
OraSession Object
Refresh Method

DragIcon Property

See Also

Applies To

Oracle Data Control.

Description

Determines the icon to be displayed as the pointer in a drag-and-drop operation.

See Visual Basic Help for more information.

See Also

Drag Method
DragDrop Event
DragMode Property
DragOver Event

DragMode Property

See Also

Applies To

Oracle Data Control.

Description

Determines manual or automatic dragging mode for a drag-and-drop operation.

See Visual Basic Help for more information.

●

See Also

Drag Method
DragDrop Event
DragIcon Property
DragOver Event

EditMode Property

See Also

Applies To

Oracle Data Control.

Description

Returns the current editing state for the current row.   Not available at design time and read-only at run time.

Usage

edit_mode = oradata1**.EditMode**

Remarks

The **EditMode** property values are:

Constant	Value	Description
ORADATA_EDITNONE	0	No editing in progress
ORADATA_EDITMODE	1	Editing is in progress on an existing row
ORADATA_EDITADD	2	A new record is being added and the copy buffer does not currently represent an actual row in the database.

These values are located in the file **ORACONST.TXT** and are intended to match similar constants in the Visual Basic file **CONSTANT.TXT**.

This property is affected only by the **Edit**, **AddNew**, and **Update** methods.

Data Type

Integer

See Also

AddNew Method
Edit Method
Update Method

Enabled Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the control can respond to user-generated events.

See Visual Basic Help for more information.

●

See Also

Visible Property

Exclusive Property

Remarks

The Oracle Data Control does not support this property.   Oracle clients generally have no control over database access.

FontBold Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the text displayed in a control is boldfaced.

See Visual Basic Help for more information.

•

See Also

FontItalics Property
FontName Property
FontSize Property
FontStrikethru Property
FontUnderline Property

FontItalics Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the text displayed in a control is italicized.

See Visual Basic Help for more information.

•

See Also

FontBold Property
FontName Property
FontSize Property
FontStrikethru Property
FontUnderline Property

FontName Property

See Also

Applies To

Oracle Data Control.

Description

Determines the font used to display text in a control.

See Visual Basic Help for more information.

See Also

FontBold Property
FontItalics Property
FontSize Property
FontStrikethru Property
FontUnderline Property

FontSize Property

See Also

Applies To

Oracle Data Control.

Description

Determines the size of the font to be used for text displayed in a control.

See Visual Basic Help for more information.

See Also

FontBold Property
FontItalics Property
FontName Property
FontStrikethru Property
FontUnderline Property

FontStrikethru Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the text displayed in a control is struck through.

See Visual Basic Help for more information.

See Also

FontBold Property
FontItalics Property
FontName Property
FontSize Property
FontUnderline Property

FontUnderline Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the text displayed in a control is underlined.

See Visual Basic Help for more information.

•

See Also

FontBold Property
FontItalics Property
FontName Property
FontSize Property
FontStrikethru Property

ForeColor Property

See Also

Applies To

Oracle Data Control.

Description

Determines the foreground color used to display text and graphics in an object.

See Visual Basic Help for more information.

•

See Also

BackColor Property

Height Property

See Also

Applies To

Oracle Data Control.

Description

Determines the height dimension of an object.

See Visual Basic Help for more information.

See Also

Left Property
Move Method
Top Property
Width Property

HelpContextID Property

Applies To

Oracle Data Control

Description

Sets the help context number for a control.

Usage

oradata1**.HelpContextID** = [numeric_expression]

Remarks

Although the Visual Basic data control does not support this property, the Oracle Data Control does.

See Visual Basic Help for more information.

Index Property

See Also

Applies To

Oracle Data Control.

Description

Specifies the number that uniquely identifies a control in a control array. Available at design time only if the control is part of a control array; read-only at run time.

See Visual Basic Help for more information.

See Also

Tag Property

Left Property

See Also

Applies To

Oracle Data Control.

Description

Determines the distance between the internal left edge of an object and the left edge of its container.

See Visual Basic Help for more information.

•

See Also

Move Method
Top Property

MousePointer Property

See Also

Applies To

Oracle Data Control.

Description

Determines the type of mouse pointer displayed when the mouse is over a particular part of a form or control at run time.

See Visual Basic Help for more information.

●

See Also

DragIcon Property
MouseMove Event

Name Property

Applies To

Oracle Data Control.

Description

Specifies the name used in code to identify a form, control, or data access object.   Not available at run time.

See Visual Basic Help for more information.

Options Property

See Also

Applies To

Oracle Data Control.

Description

Determines one or more characteristics of the database and all dynasets associated with the data control. Read/write at design time and run time.

Usage

oradata1**.Options** = *database_options*
database_options = oradata1**.Options**

Remarks

This property is a bit flag word used to set the optional modes of the database. If *options* = 0, the default settings will apply. The following modes are available:

Column Defaulting mode.
The default mode is called VB mode. In VB mode, field (column) values not explicitly set are set to NULL when using **AddNew** or **Edit**.
Optionally you can use Oracle mode. Oracle mode indicates that changes made to fields (columns) are immediately reflected in the local mirror by retrieving the changed row from the database, thus allowing Oracle to set defaults for the columns and perform required calculations.
Column Defaulting mode affects the behavior of the **AddNew** and **Edit** methods.

Lock Wait mode.
The default mode is called Wait mode. In Wait mode, when dynaset rows are about to be modified (using **Edit**), the existing row in the database is retrieved using SELECT ... FOR UPDATE to lock the row in the database. If the row about to be changed has been locked by another process (or user), the SELECT ... FOR UPDATE, waits until the row is unlocked before proceeding.
Optionally you may use NoWait mode. NoWait mode results in an immediate return of an error code, indicating that the row about to be updated is locked.
Lock Wait mode also affects any SQL statements processed using **ExecuteSQL**.

The options flag values are:

Constant	**Value**	**Description**
ORADB_DEFAULT	&H0&	Accept the default behavior.
ORADB_ORAMODE	&H1&	Let Oracle set default field (column) values.
ORADB_NOWAIT	&H2&	Do not wait on row locks when executing a "SELECT ... FOR UPDATE".

Options may be combined by adding their respective values.

These values can be found in the file **ORACONST.TXT**.

This property is the same as the options passed to the **OpenDatabase** method.   Just as with **OpenDatabase**, these options affect the **OraDatabase** object and <u>all</u> associated dynasets created from that database.

Changing this property does not take effect until a **Refresh** method is sent to the data control.

Data Type

Long Integer

See Also

AddNew Method
Edit Method
CreateDynaset Method
OpenDatabase Method
OraDatabase Object
OraDynaset Object
Refresh Method
Locks and Editing

ReadOnly Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether the dynaset will be used for read-only operations. Read/write at design time and run time.

Usage

oradata1**.ReadOnly** = [**<u>True</u>** | **False**]

Remarks

By default, **ReadOnly** is **False** which means that an attempt will be made to create an updatable dynaset by selecting ROWIDs from the database. If **ReadOnly** is set to **True**, a nonupdatable dynaset is created (ROWIDs are not selected from the database and cached) and operations will be somewhat faster.

If the SELECT statement contains a LONG or LONG RAW column, ROWIDs are needed whether the dynaset will be updatable or not.

Changing this property does not take effect until a **Refresh** method is sent to the data control.

Data Type

Integer (Boolean)

See Also

CreateDynaset Method
Long and Long Raw Columns
Refresh Method

Recordset Property

See Also

Applies To

Oracle Data Control.

Description

Returns a dynaset defined by the data control's **Connect**, **DatabaseName**, and **RecordSource** properties.   Not available at design time and read-only at run time.

Usage

Set *oradynaset* = oradata1**.RecordSet**

Remarks

The properties and methods of this dynaset are the same as those of any other dynaset object.

Data Type

OLE Object (OraDynaset)

See Also

Connect Property
DatabaseName Property
MoveFirst, MoveLast, MoveNext, MovePrevious Methods
OraDynaset Object
OraFields Collection
OraParameters Collection
RecordSource Property

RecordSource Property

See Also　　　　Example

Applies To

Oracle Data Control.

Description

The SQL select statement to be used to create the data controls **RecordSet**. Read/write at design time and run time.

Usage

oradata1**.RecordSource** = [SQL SELECT Statement]

Remarks

The SQL statement must be a SELECT statement; otherwise an error is returned.   Features such as views, synonyms, column aliases, schema references, table joins, nested selects, and remote database references can be used freely; object names are not modified in any way.

The updatability of the resultant dynaset depends on the Oracle SQL rules of updatability, on the access you have been granted, and on the ReadOnly property. In order to be updatable, three conditions must be met:

1. the SQL statement must refer to a simple column list or to the entire column list (*),
2. the SQL statement must not set the read-only flag of the options argument, and
3. Oracle must permit ROWID references to the selected rows of the query.

Any SQL statement that does not meet these criteria is processed, but the results are not updatable and the dynasets **Updatable** property returns **False***.*

Changing this property does not take effect until a Refresh method is sent to the data control.

You can use SQL bind variables in conjunction with the **OraParameters** collection.

If this property is NULL or empty, then an **OraDynaset** object is ***not*** created, but **OraSession**, **OraConnection**, and **OraDatabase** objects are created for the data control.   This behavior enables access to these objects prior to creation of a dynaset.   For example, a NULL **RecordSource** might be used to instantiate the database object for the purpose of adding parameters. **RecordSource** can then be set at run time, making use of the automatic binding of database parameters.

Data Type

String

See Also

Connect Property
DatabaseName Property
OraConnection Object
OraDatabase Object
OraDynaset Object
OraParameters Collection
OraSession Object
RecordSet Property
Refresh Method
Updatable Property

RecordSource Property Example

This example demonstrates the use of SQL bind variables (parameters) in the **RecordSource** property of the data control. Copy this code into the definition section of a form containing a data control named oradata1. Then press F5.

```
Sub Form_Load ()

 'Set the username and password.
 oradata1.Connect = "scott/tiger"

 'Set the databasename.
 oradata1.DatabaseName = "ExampleDb"

 'Refresh the data control without setting the
 ' recordsource. This has the effect of creating
 ' the underlying database object so that parameters
 ' may be added.
 oradata1.Refresh

 'Set the recordsource and use a SQL parameter.
 oradata1.RecordSource = "select * from emp where job = :job"

 'Add the job input parameter with initial value MANAGER.
 oradata1.Database.Parameters.Add "job", "MANAGER", 1

 'Refresh the data control.
 'Only employees with the job MANAGER will be contained
 'in the dynaset.
 oradata1.Refresh

 'Change the value of the job parameter to SALESMAN.
 oradata1.Database.Parameters("job").Value = "SALESMAN"

 'Refresh ONLY the recordset.
 'Only employees with the job SALESMAN will be contained
 ' in the dynaset.
 oradata1.Recordset.DbRefresh

End Sub
```

Session Property

See Also

Applies To

Oracle Data Control.

Description

The session object associated with the data control.   Not available at design time and read-only at run time.

Usage

orasession = oradata1**.Session**

Remarks

This property is equivalent to referencing oradata1.**Database**.**Session**.   If the data control has not been refreshed, any references to this property result in an "Object variable not set" runtime error.

Data Type

OLE Object (**OraSession**)

See Also

OraDatabase Object
OraSession Object
OraSessions Collection

Tag Property

Applies To

Oracle Data Control.

Description

Stores any extra data needed by your application.

See Visual Basic Help for more information.

Top Property

See Also

Applies To

Oracle Data Control.

Description

Determines the distance between the internal top edge of an object and the top edge of its container.

See Visual Basic Help for more information.

•

See Also

Move Method
Left Property

TrailingBlanks Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether trailing blanks should be stripped from character string data retrieved from the database.   Read/write at design time and run time.

Usage

oradata1**.TrailingBlanks** = [**True | False**]

Remarks

By default, **TrailingBlanks** is **False**   This means that trailing blanks will be stripped from character string data retrieved from the database.

Changing this property has no effect until a **Refresh** method is sent to the data control.

Data Type

Integer (Boolean)

See Also

CreateDynaset Method
Refresh Method

Visible Property

See Also

Applies To

Oracle Data Control.

Description

Determines whether an object is visible or hidden.

See Visual Basic Help for more information.

•

See Also

Enabled Property

Width Property

See Also

Applies To

Oracle Data Control.

Description

Determines the width dimension of an object.

See Visual Basic Help for more information.

See Also

Height Property
Left Property
Move Method
Top Property

Drag Method

See Also

Applies To

Oracle Data Control.

Description

Begins, ends, or cancels dragging controls.

See Visual Basic Help for more information.

•

See Also

DragDrop Event
DragIcon Property
DragMode Property
DragOver Event
MousePointer Property

Move Method

See Also

Applies To

Oracle Data Control.

Description

Moves a form or control.

See Visual Basic Help for more information.

•

See Also

Height Property
Left Property
Top Property
Width Property

Refresh Method

See Also

Applies To

Oracle Data Control.

Description

This method re-creates the **OraDatabase** and **OraDynaset** objects referenced within the data control and reestablishes a dynaset using the SQL statement from the **RecordSource** property and the connection information from the **Connect** and **DatabaseName** properties.

Usage

oradata1**.Refresh**

Remarks

If an existing dynaset has been assigned to an object variable in Visual Basic, then **Refresh** creates a new dynaset for the data control, but the old dynaset continues to be available for use until all references to it are removed.

See Also

Connect Property
DatabaseName Property
OraDatabase Object
OraDynaset Object
RecordSource Property
SQL Property

UpdateControls Method

See Also Example

Applies To

Recordset property of the Oracle Data Control.

Description

Gets the current record from a data control's recordset and displays the appropriate data in controls bound to that data control.

Usage

oradata1**.**Recordset**.UpdateControls**
oradata1**.**Recordset**.DbUpdateControls** (Required for Visual Basic 3.0 users)

Remarks

Use this method to allow the user to cancel changes made to bound controls and restore the contents of those controls to their original values.

This method has the effect of making the current record current again, except that no events occur.

Note: Due to limitations of Custom Controls under Visual Basic 3.0, this method cannot be made available directly from the data control. Rather, it must be provided as a method of the data control's Recordset.

See Also

Recordset Property
UpdateRecord Method

UpdateControls Method Example

This example cancels changes made to bound controls and restores the data to the original values.   Copy this code into the definition section of a form that has a data control named oradata1 (which has been successfully refreshed) and has the KeyPreview property set to **True**.   Then press F5.

```
Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)
      Const KEY_ESCAPE = &H1B
      If KeyCode = KEY_ESCAPE Then
            oradata1.recordset.DbUpdateControls
      End If
End Sub
```

UpdateRecord Method

See Also

Applies To

Recordset property of the Oracle Data Control.

Description

Saves the current values of bound controls.

Usage

oradata1**.**Recordset**.UpdateRecord**
oradata1**.**Recordset**.DbUpdateRecord** (Required for Visual Basic 3.0 users)

Remarks

This method allows you to save the current value of bound controls during a **Validate** event without generating another **Validate** event.

This method has the effect of executing the **Edit** method, changing a field, and executing the **Update** method, except that no events occur.

Note: Due to limitations of Custom Controls under Visual Basic 3.0, this method cannot be made available directly from the data control. Rather, it must be provided as a method of the data control's Recordset.

See Also

Edit Method
Recordset Property
Update Method
Validate Event

ZOrder Method

Applies To

Oracle Data Control.

Description

Places a specified form or control at the front or back of the z-order within its graphical level.

See Visual Basic Help for more information.

DragDrop Event

See Also

Applies To

Oracle Data Control.

Description

Occurs when a drag-and-drop operation is completed as a result of either dragging a control over a form or control and releasing the mouse button, or using the **Drag** method with its action argument = 2 (Drop).

See Visual Basic Help for more information.

See Also

Drag Method
DragIcon Property
DragMode Property
DragOver Event
MouseDown Event
MouseMove Event
MouseUp Event

DragOver Event

See Also

Applies To

Oracle Data Control.

Description

Occurs when a drag-and-drop operation is in progress.   You can use this event to monitor when the mouse pointer enters, leaves, or is directly over a valid target.   The mouse pointer position determines which target object receives this event.

See Visual Basic Help for more information.

•

See Also

Drag Method
DragDrop Event
DragIcon Property
DragMode Property
MouseDown Event
MouseMove Event
MouseUp Event

Error Event

See Also

Applies To

Oracle Data Control.

Description

This event is fired whenever an interactive operation causes an error.   You can perform some operations directly with the data control, such as using the data control buttons or when the data control refreshes automatically when the form loads.   In these cases, the **Error** event is fired instead of causing a normal runtime error.

See Visual Basic Help for more information.

•

See Also

AddNew Method
Delete Method
MoveFirst, MoveLast, MoveNext, MovePrevious Methods

MouseDown Event

See Also

Applies To

Oracle Data Control.

Description

This event is fired whenever a mouse button is pressed (**MouseDown**) and the mouse pointer is over the data control, or has been *captured* by the data control. The mouse is *captured* if a mouse button has been pressed previously over the data control until all corresponding **MouseUp** events have been received.

See Visual Basic Help for more information.

See Also

MouseMove Event
MousePointer Property
MouseUp Event

MouseMove Event

See Also

Applies To

Oracle Data Control.

Description

This event is fired continuously whenever the mouse pointer moves across the data control.   Unless another object has not *captured* the mouse, the data control recognizes a **MouseMove** event whenever the mouse position is within its borders.

See Visual Basic Help for more information.

See Also

MousePointer Property
MouseDown Event
MouseUp Event

MouseUp Event

See Also

Applies To

Oracle Data Control.

Description

This event is fired whenever a mouse button is released (**MouseUp**) and the mouse pointer is over the data control, or has been *captured* by the data control.   The mouse is *captured* if a mouse button has been pressed previously over the data control until all corresponding **MouseUp** events have been received.

See Visual Basic Help for more information.

See Also

MouseMove Event
MousePointer Property
MouseDown Event

Reposition Event

See Also

Applies To

Oracle Data Control.

Description

This event is fired whenever the database record pointer is successfully repositioned to a new location.   The **Validate** event is always fired before **Reposition**.

See Visual Basic Help for more information.

See Also

Error Event
FindFirst, FindLast, FindNext, FindPrevious Methods
MoveFirst, MoveLast, MoveNext, MovePrevious Methods
Validate Event

Validate Event

See Also

Applies To

Oracle Data Control.

Description

This method is called whenever a variety of circumstances occur. It is sent when an attempt is made to move to a new record position, to delete a record, add a record, move to a bookmark, or to roll back the dynasets in the session. **Validate** is always called before the operation proceeds and any action is taken.

See Visual Basic Help for more information.

See Also

AddNew Method
BookMark Property
Close Method
Delete Method
Edit Method
EditMode Property
FindFirst, FindLast, FindNext, FindPrevious Methods
MoveFirst, MoveLast, MoveNext, MovePrevious Methods
Update Method
UpdateRecord Method