Oracle® Objects for OLE

Developer's Guide

11*g* Release 2 (11.2) for Microsoft Windows

E12245-01

February 2010

ORACLE®

Oracle Objects for OLE Developer's Guide, 11*g* Release 2 (11.2) for Microsoft Windows

E12245-01

Primary Authors: Janis Greenberg, Christian Shay

Contributing Authors: Riaz Ahmed, Kiminari Akiyama, Steven Caminez, Naveen Doraiswamy, Neeraj Gupta, Sinclair Hsu, Alex Keh, Chithra Ramamurthy, Ashish Shah, Martha Woo

Contents

3 Basic Features

4 Advanced OO4O Features

5 Tuning and Troubleshooting

6 Quick Tour with Visual Basic

7 Code Wizard for Stored Procedures

8 Introduction to Automation Objects

9 Server Objects

10 Server Methods

11 Server Properties

Preface

This document explains how to install, configure, and use Oracle Objects for OLE (OO4O). It covers features of Oracle Database that apply to Microsoft Windows operating systems.

Oracle Objects for OLE (OO4O) allows easy access to data stored in Oracle databases with any programming or scripting language that supports the Microsoft COM Automation.

Audience

Oracle Objects for OLE Developer's Guide is intended for programmers developing applications to access an Oracle database using Oracle Objects for OLE. This documentation is also valuable to systems analysts, project managers, and others interested in the development of database applications.

To use this document, you must have a working knowledge of application programming using Visual Basic or Microsoft C/C++ and knowledge of Component Object Model (COM) concepts.

Readers should also be familiar with the use of structured query language (SQL) to access information in relational database systems.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

For more information, see these Oracle resources:

- *Oracle Objects for OLE C++ Class Library Developer's Guide*, available as online help
- *Oracle Services for Microsoft Transaction Server Developer's Guide*
- *Oracle Database Platform Guide for Windows*
- *Oracle Database Concepts*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database Reference*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database SecureFiles and Large Objects Developer's Guide*
- *Oracle Database Object-Relational Developer's Guide*
- *Oracle Streams Advanced Queuing User's Guide*
- *Oracle XML DB Developer's Guide*
- *Oracle XML Developer's Kit Programmer's Guide*
- *Oracle Database PL/SQL User's Guide and Reference*
- *Oracle Net Services Reference Guide*
- *Oracle Database Globalization Support Guide*
- *Oracle Database Oracle Real Application Clusters Administration and Deployment Guide*

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

http://www.oracle.com/technology/contact/welcome.html

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

http://www.oracle.com/technology/documentation/index.html

For additional information, see:

http://www.microsoft.com

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Objects for OLE?

This section describes new features of Oracle Database 11*g* Release 2 (11.2) and provides pointers to additional information. New features information from previous releases is also retained to help those users migrating to the current release.

The following sections describe the new features in Oracle Oracle Objects for OLE:

Oracle Database 11*g* Release 1 (11.1) and Release 2 (11.2) New Features

There are no new features for these releases.

Oracle Database 10*g* Release 2 (10.2) New Features

There are no new features for this release.

Documentation for Oracle Objects for OLE was improved and reorganized, although there is no additional content. The documentation was reformatted to a printable, PDF format. PDF and HTML are provided in the Documentation Library. Online Help in WinHelp format is no longer provided.

Oracle Database 10*g* Release 1 (10.1) New Features

- Support for Oracle Grid Computing

  Oracle Objects for OLE is grid-enabled, allowing developers to take advantage of Oracle database grid support without having to make changes to their application code.

- Support for New Data Types

  Oracle Objects for OLE provides support for the `BINARY_DOUBLE` and `BINARY_FLOAT` data types introduced in Oracle Database 10*g*. Instances of these types can be fetched from the database or passed as input or output variables to SQL statements and PL/SQL locks, including stored procedures and functions.

  The following constants were added in the `oraconst.txt` to bind the `BINARY_DOUBLE` and `BINARY_FLOAT` data types.

  – `ORATYPE_BDOUBLE`, Oracle data type `BINARY_DOUBLE`, value `101`

- `ORATYPE_BFLOAT`, Oracle data type `BINARY_FLOAT`, value `100`

- Support for Multiple Oracle Homes

  Oracle Objects for OLE can be installed in multiple Oracle homes, starting with release 10.1. However, being a Component Object Model (COM) component, only one instance can be active on the computer. This means that the current (latest) installation renders the previous one inactive.

  To make multiple Oracle homes available, the use of a `KEY_`*`HOMENAME`* is required. Also, some of the Oracle Objects for OLE files include a version number.

  See Also:

  - "Installation" on page 1-5
  - "Oracle Objects for OLE Redistributable Files" on page 1-6
  - "Tuning and Customization" on page 5-2 for information about the Windows registry subkey

1

Introducing Oracle Objects for OLE

This chapter introduces Oracle Objects for OLE (OO4O).

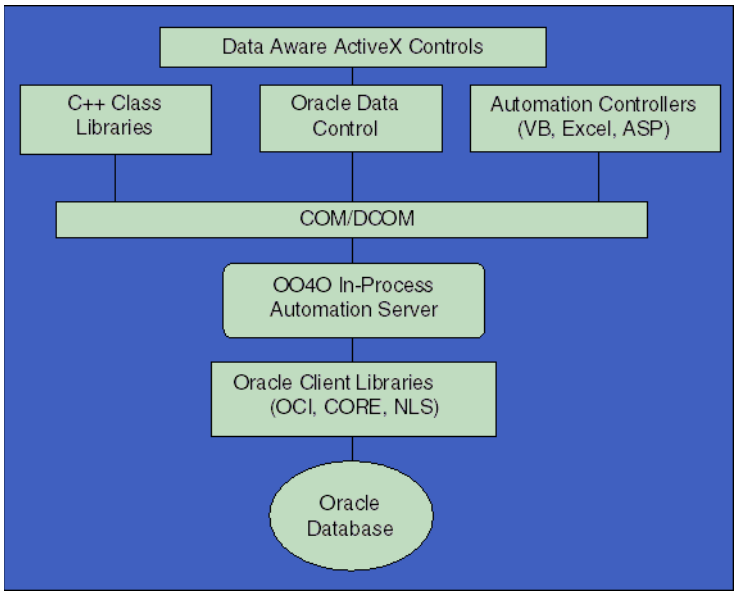
This chapter contains these topics:

Overview of Oracle Objects for OLE

Oracle Objects for OLE (OO4O) allows you to access data stored in Oracle databases with any programming or scripting language that supports Microsoft COM Automation and ActiveX technology. This includes Visual Basic, Visual C++, Visual Basic for Applications (VBA), IIS Active Server Pages (VBScript and JavaScript), and others.

Figure 1–1 illustrates the software layers that comprise the OO4O product.

Figure 1–1 Software Layers of OO4O



OO4O provides the following:

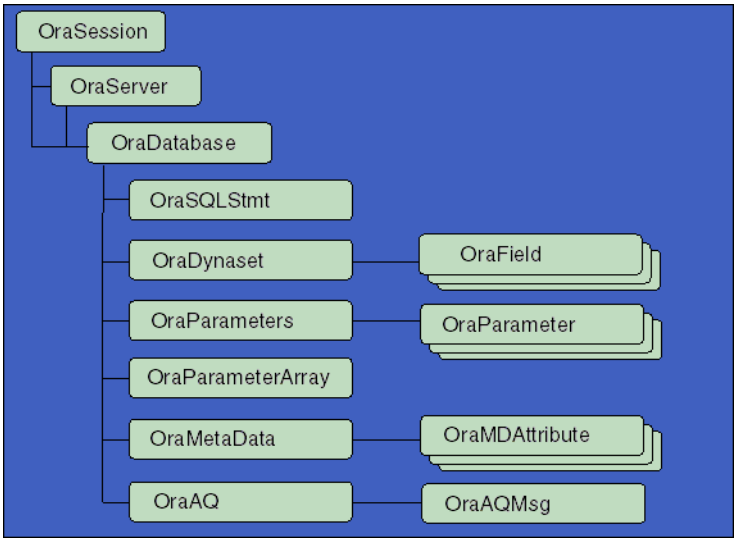
- Oracle Objects for OLE In-Process Automation Server
- Oracle Data Control
- Oracle Objects for OLE C++ Class Library

Oracle Objects for OLE In-Process Automation Server

The OO4O In-Process Automation Server is a set of COM Automation Objects for connecting to Oracle databases, executing SQL statements and PL/SQL blocks, and accessing the results.

Figure 1–2 illustrates the object model that comprise the OO4O product.

Figure 1–2 Automation Objects



Unlike other COM-based database connectivity APIs, such as Microsoft ActiveX Data Objects (ADO), the OO4O Automation Server was created specifically for use with Oracle databases. It provides an optimized API for accessing features that are unique to the Oracle database and are otherwise cumbersome or unavailable from ODBC or OLE DB components.

OO4O provides key features for accessing Oracle databases efficiently and easily in environments ranging from the typical two-tier client/server applications, such as those developed in Visual Basic or Excel, to application servers deployed in multitiered application server environments such as Web server applications in Microsoft Internet Information Server (IIS) or Microsoft Transaction Server (MTS).

See Also: "Introduction to Automation Objects" on page 8-1

Features include:

- Oracle 11*g* is grid enabled, allowing developers to take full advantage of grid support without changes being required to existing code.
- Tunable client-side, scrollable and updatable cursors for easy and efficient access to result sets of queries.
- PL/SQL support for execution of PL/SQL anonymous blocks and stored procedures. This includes support for the Oracle data types, such as PL/SQL cursors, that are needed for parameters of PL/SQL stored procedures.
- Support for array fetches, updates, and inserts resulting in reduced network round-trips.
- Connection pooling to allow development of scalable middle tier application components, such as IIS Active Server Pages, that use and serve dynamic content stored in Oracle databases.
- Support for COM+ and Microsoft Transaction Server (MTS) co-ordinated transactions.
- Seamless access to instances of advanced Oracle data types:
  - Object references (REFs)
  - Object instances (Objects)
  - Nested tables
  - `VARRAY`s
  - `BLOB`s, `CLOB`s, `NCLOB`s, and `BFILE`s
- XML generation.
- Full support for advanced queuing.
- Support for publishing, detecting, and subscribing to database events.
- Support for asynchronous processing of SQL statements and PL/SQL blocks.
- Easy to use interface for describing schema objects.
- The Oracle code wizard for stored procedures, which automatically generates OO4O code to execute PL/SQL or Java stored procedures.
- Thread safety, allowing safe access to automation objects in multithreaded environments.

See Also:

- Chapter 9, "Server Objects"
- Chapter 11, "Server Properties"
- Chapter 10, "Server Methods"
- Chapter 4, "Advanced OO4O Features"

Oracle Data Control

Oracle Data Control is an ActiveX control that is designed to simplify the exchange of data among an Oracle database and visual controls such as edit, text, list, and grid controls in Visual Basic and other development tools that support custom controls.

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

A data control enables you to bind it to other controls that display a field, a record, or multiple records of the underlying dynaset. When record movement occurs, data in bound controls stay synchronized with the current record of the dynaset. If a user changes data in a control that is bound to a data control, the changes are automatically reflected in the underlying dynaset and database.

The Oracle Data Control is compatible with the Microsoft data control included with Visual Basic. If you are familiar with the Visual Basic data control, learning to use Oracle Data Control is quick and easy. Communication between data-aware controls and a Data Control is governed by a protocol specified by Microsoft.

See Also:

- "Setting Oracle Data Control Properties with the Properties Window" on page 2-9
- "Setting Oracle Data Control Properties Programmatically" on page 2-11
- "Data Control Events" on page 12-1

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 OO4O Automation 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. This library helps C++ developers avoid writing COM client code to access the OO4O interfaces.

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 of a table in the database). The bound class supports late, run-time binding, as is available in Visual Basic. The Oracle Objects for OLE C++ Class Library is supported for Microsoft Visual C++ and the Microsoft Foundation Classes for the bound class.

See Also: *Oracle Objects for OLE C++ Class Library Developer's Guide* available as online help

Required Setups

This section discusses the required setups for using Oracle Objects for OLE.

Installation

Oracle Objects for OLE can be installed in multiple Oracle homes, starting with Oracle Database 10*g*. However, being a COM component, only one instance can be active on the computer. This means that the current (latest) installation renders the previous one inactive. You can switch Oracle homes by using the Oracle installer.

System Requirements

The following system requirements are necessary to install Oracle Objects for OLE:

- Windows Operating System:
  - 32-bit: Windows 7 (Professional, Enterprise, and Ultimate Editions), Windows Vista (Business, Enterprise, and Ultimate Editions), Windows Server 2008 (Standard, Enterprise, Datacenter, Web, and Foundation Editions), Windows Server 2003 R2 (all editions), Windows Server 2003 (all editions), or Windows XP Professional Edition.

    Oracle supports 32-bit Oracle Objects for OLE on x86, AMD64, and Intel EM64T processors on these operating systems.
- A local or remote Oracle database (Oracle9*i* Release 2 or higher)
- Oracle Client. Oracle Universal Installer ensures that the RSFs are installed as part of the OO4O installation.
- The OO4O automation server requires an application that supports COM Automation such as:
  - Microsoft Visual Basic
  - Microsoft Excel
  - Microsoft Access
  - Microsoft Internet Information Server (IIS)
  - Borland Delphi

Other Requirements

The following other requirements may be necessary:

- The Oracle Data Control requires Visual Basic.
- The Oracle Objects for OLE C++ Class Library requires Microsoft Visual C++ Version 6.0 or later.
- The OO4O Code Wizard requires Visual Basic 6. Visual Basic 6 must be installed before installing the Code Wizard.
- The Oracle In-Process Server Type library (`oip`*`VER`*`.tlb`) must be referenced when an OO4O Visual Basic project is developed.

  To do this, select **References** from the Project menu (VB 5.0/6.0) and check the box next to the Oracle In-Process Server 5.0 Type Library, which should be pointing to the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\bin\oip`*`VER`*`.tlb` file. See "Using Oracle Objects for OLE Automation with Visual Basic" on page 2-2 for detailed information.

See Also: "Oracle Objects for OLE Redistributable Files" on page 1-6 and "Troubleshooting" on page 5-14 for further information on Oracle Objects for OLE requirements

Oracle Objects for OLE File Locations

As part of the OO4O installation, the following directories are created and contain the corresponding files:

- *ORACLE_BASE**ORACLE_HOME*\OO4O - SQL scripts and constants file
- *ORACLE_BASE**ORACLE_HOME*\OO4O\CPP - Libraries, include files, DLLs, and source for the class library
- *ORACLE_BASE**ORACLE_HOME*\OO4O\CPP\MFC - Libraries, include files, and source for the MFC Bound Class Library
- *ORACLE_BASE**ORACLE_HOME*\OO4O\EXCEL\SAMPLES - Excel samples
- *ORACLE_BASE**ORACLE_HOME*OO4O\VB\SAMPLES - Visual Basic samples
- *ORACLE_BASE**ORACLE_HOME*\OO4O\VB\SAMPLES\QT - Visual Basic Quick Tour guide
- *ORACLE_BASE**ORACLE_HOME*\OO4O\IIS\SAMPLES - IIS samples
- *ORACLE_BASE**ORACLE_HOME*\OO4O\codewiz - OO4O Code Wizard samples

Component Certifications

Find the latest certification information at My Oracle Support (formerly Oracle*MetaLink*):

http://metalink.oracle.com/

You must register online before using My Oracle Support. After logging into My Oracle Support, select **Product Lifecycle** from the left column. From the Products Lifecycle page, click **Certifications**. Other Product Lifecycle options include Product Availability, Desupport Notices, and Alerts.

Oracle Objects for OLE Redistributable Files

This section discusses files that can be redistributed or updated on a computer that belongs to an end user or a developer.

If you cannot guarantee that your end users have the current release of Oracle Objects for OLE installed on their computers, you need to redistribute specific files that are part of Oracle Objects for OLE along with your OO4O application. A typical scenario might be if OO4O is installed as a patch without use of Oracle Universal Installer.

Redistributable File Locations

Table 1–1 lists the Redistributable file locations with comments and further actions that are needed.

Table 1–1 Redistributable File Locations

Files	Place in Directory	Further Actions	Comments
oip*VER*.dll	*ORACLE_BASE*\ *ORACLE_HOME*\bin	Execute the following from a command prompt: *drive:**path*> regsvr32.exe oip*VER*.dll	None.
oip*VER*.tlb	*ORACLE_BASE*\ *ORACLE_HOME*\bin	None.	None.
oraansi*VER*.dll	*ORACLE_BASE*\ *ORACLE_HOME*\bin	None.	Change *VER* to the current version.
oo4oparm.reg	*ORACLE_BASE*\ *ORACLE_HOME*\oo4o	Edit for the correct *ORACLE_HOME* location and HOME*ID* on your computer. Execute the following from a command prompt: *drive:**path*> oo4oparm.reg	File provided to register OO4O configuration information.
oiplang.msb	*ORACLE_BASE*\ *ORACLE_HOME* \oo4o\mesg	None.	This message file is language-specific. oipus.msb is the English version, and oipja.msb is the Japanese version.
oraclm32.dll (for Microsoft VC++) or oradc.ocx	*ORACLE_BASE*\ *ORACLE_HOME*\bin	For oradc.ocx, execute: regsvr32.exe oradc.ocx *drive:**path*>	Distribute the files that correspond to the development software used in your application.

Additionally, ensure that the system requirements described in "Overview of Oracle Objects for OLE" on page 1-1 are met.

You must also distribute the files from the following list that correspond to the development software you used to build your application:

- oraclm32.dll (for Microsoft Visual C++)
- oradc.ocx

Updating Files and Registrations

The oo4oparm.reg file is provided to register OO4O configuration information. Review this file and edit it as necessary to reflect the correct *ORACLE_HOME* location and HOME*ID* on your computer. To register oip*VER*.dll and enter the OO4O configuration information for oo4oparm.reg in the registry, execute the following from a command prompt:

drive:*path*> regsvr32.exe oip*VER*.dll

drive:*path*> oo4oparm.reg

The message file oiplang.msb should also be provided and copied to the *ORACLE_BASE**ORACLE_HOME*\oo4o\mesg directory. The message file is specific to a language. For example, oipus.msb is the English version and oipja.msb is the Japanese version.

Note: Oracle Data Control (`oradc.ocx`) must be registered to function. The OLE Control Extension (OCX) can be registered by executing the following at the command prompt:

```
drive:\path> regsvr32.exe oradc.ocx
```

See Also: "Oracle Data Control" on page 1-4

2

Using Oracle Objects for OLE with Automation Clients

This chapter describes the use of automation clients to access Oracle data.

This chapter contains these topics:

- Using Automation Clients Overview
- Demonstration Schema and Code Examples
- Using Oracle Objects for OLE Automation with Visual Basic
- Using OO4O Automation with Active Server Pages (ASP)
- Using Oracle Objects for OLE Automation with Excel
- Using Microsoft C++
- Using Oracle Data Control with Visual Basic
- Using the Oracle Data Control with MS Visual C++

Using Automation Clients Overview

Oracle Objects for OLE (OO4O) is designed to provide quick and efficient access to the data in an Oracle database using various programming or scripting languages.

OO4O can be easily used with Visual Basic, Excel, Active Server Pages, Internet Information Server (IIS), and other development tools.

Oracle Data Control with Visual Basic allows another method of accessing Oracle data.

Examples are provided for specific methods and properties in this developer's guide. Additionally, example programs are installed with Oracle Objects for OLE and are located in the *ORACLE_BASE**ORACLE_HOME*\oo4o\ directory under VB, EXCEL, IIS, CPP, and so on.

A Quick Tour of OO4O with Visual Basic is also provided.

See Also: Chapter 6, "Quick Tour with Visual Basic"

Demonstration Schema and Code Examples

The code examples included in this developer's guide and the example applications shipped with Oracle Objects for OLE are designed to work with a demonstration schema (database tables and other objects) and a demonstration user and password, `scott/tiger`. Code examples are located in the *ORACLE_BASE**ORACLE_HOME*\oo4o directory.

Demonstration Schema Creation

You can create the OO4O demonstration schema with the `demobld7.sql` script located in the *ORACLE_BASE*`\`*ORACLE_HOME*`\oo4o` directory. You can drop the demonstration schema with the `demodrp7.sql` script.

Demonstration Schema

The demonstration schema includes the following references:

- Demonstration tables `EMP` and `DEPT`.
- The user `scott` with password `tiger` (`scott/tiger`).
- The network alias, *`ExampleDb`*.

  Refer to *Oracle Net Services Administrator's Guide* for assistance in setting up the network service (database) alias and the `tnsnames.ora` file.

  In many of the examples, you can access a local database using `" "` (a null string) for the network alias.

Other Schemas

Occasionally other schemas are required to run examples. The introductions to the examples contain names and locations of the schemas (in the appendix).

See Also: "Additional Schemas" on page A-2

Related Files

The *ORACLE_BASE*`\`*ORACLE_HOME*`\oo4o` directory contains the following items:

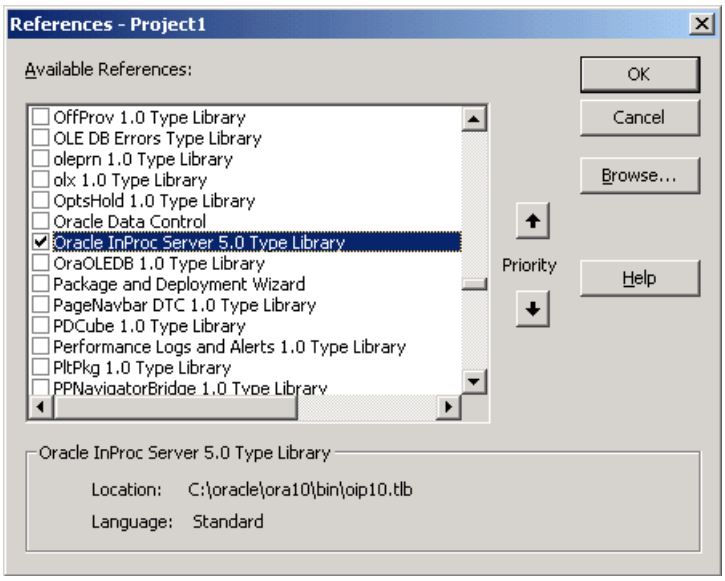
- OO4O example programs.

  Subdirectories contain both C++ and Visual Basic examples.
- The `oraexamp.sql` script, used to create stored procedures. Additional scripts, such as `multicur.sql` and `empcur.sql`, are provided to set up other example programs.
- Oracle Objects for OLE global constant file, `oraconst.txt`, which contains constant values used for option flags and property values. This file is usually not needed as these constants are also included with the Oracle In-Process Server type library.

Using Oracle Objects for OLE Automation with Visual Basic

This example contains code fragments that demonstrate how to create all objects required by a dynaset and then create the dynaset itself.

1. Start Visual Basic and create a new project. From the **Project** menu, select **References** and check **InProcServer 5.0 Type Library**.



2. Start Visual Basic and create a new project. Then, add the following code to the Declarations section of a form:

```
...
' Declare variables
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim OraFields As OraFields
```

3. Add the following code to the load procedure associated with the form to display the Oracle data:

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

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

' You can now display or manipulate the data in the dynaset. For example:
Set OraFields = OraDynaset.fields
OraDynaset.movefirst
Do While Not OraDynaset.EOF
MsgBox OraFields("ename").Value
OraDynaset.movenext
Loop
End Sub
```

4. Run the form to view the results.

See Also:

- "Using Oracle Objects for OLE with Automation Clients" on page 6-1 for more information about using OO4O with Visual Basic
- OraClient Object on page 9-18
- OraDatabase Object on page 9-28
- OraDynaset Object on page 9-30
- OraField Object on page 9-33
- OraParameter Object on page 9-50
- OraSession Object on page 9-58

Using OO4O Automation with Active Server Pages (ASP)

This example uses Active Server Pages (ASP) in a Microsoft Internet Information Server (IIS) to demonstrate the connection pooling feature of Oracle Objects for OLE. The sample code executes a SQL `SELECT` query and returns the result as an HTML table. The database connection used in this script is obtained from a pool that is created when the `global.asa` file is executed.

To use Oracle Objects for OLE with OLE Automation and IIS, you need to install IIS 3.0 or later, including all ASP extensions. On the computer where IIS is running, an Oracle database must also be accessible.

Note: The sample code for this example is available in the *ORACLE_BASE*`\`*ORACLE_HOME*`\oo4o\iis\samples\asp\connpool` directory.

---

1. Start SQL*Plus and log in to the Oracle database as `scott/tiger`.

   Create the following PL/SQL procedures:

```
-- creates PL/SQL package to be used in ASP demos
create or replace package ASP_demo as
    --cursor c1 is select * from emp;
    type empCur is ref cursor;
PROCEDURE GetCursor(p_cursor1 in out empCur, indeptno IN NUMBER,
                  p_errorcode OUT NUMBER);
END ASP_demo;
/
```

   Create or replace the ASP_demo package body as follows:

```
PROCEDURE GetCursor(p_cursor1 in out empCur, indeptno IN NUMBER,
                p_errorcode OUT NUMBER) is
BEGIN
    p_errorcode:= 0;
    open p_cursor1 for select * from emp where deptno = indeptno;
EXCEPTION
    When others then
    p_errorcode:= SQLCODE;
END GetCursor;
...
END ASP_demo;
/
```

2. Create the Active Server Pages (ASP) sample code. The OO4O related code is in **bold**.

```
'GLOBAL.ASA

<OBJECT RUNAT=Server SCOPE=Application ID=OraSession
             PROGID="OracleInProcServer.XOraSession"></OBJECT>
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart

'Get an instance of the Connection Pooling object and
'create a pool of OraDatabase
OraSession.CreateDatabasePool 1,40,200,"exampledb", "scott/tiger",  0
End Sub

'OO4ODEMO.ASP

<html>
<head>
<title>Oracle Objects For OLE (OO4O) </title>
</head>
<body BGCOLOR="#FFFFFF">
<font FACE="ARIAL,HELVETICA">
<h2 align="center">Oracle Objects For OLE (OO4O) </h2>
<form ACTION="OO4ODEMO.asp" METHOD="POST">
<%
SqlQuery = Request.Form("sqlquery")
%>
<p>This sample executes a SQL SELECT query and returns the result as an HTML
table. The database connection used in this script is obtained from a pool that
is created when the <strong>global.asa</strong> is executed. </p>
<p>SQL Select Query: <input SIZE="48" NAME="sqlquery"> </p>
<p><input TYPE="SUBMIT"> <input TYPE="RESET"> <input LANGUAGE="VBScript"
TYPE="button" VALUE="Show ASP Source" ONCLICK="Window.location.href =
&quot;oo4oasp.htm&quot;"

NAME="ShowSrc"></p>
</form>
<%
If SqlQuery = "" Then
%>
<% Else %>
<table BORDER="1">
<%
Set OraDatabase = OraSession.GetDatabaseFromPool(10)
Set OraDynaset = OraDatabase.CreateDynaset(SqlQuery,0)
Set Columns = OraDynaset.Fields
%>
<tr>
<td><table BORDER="1">
<tr>
<% For i = 0 to Columns.Count - 1 %>
<td><b><% = Columns(i).Name %></b></td>
<% Next %>
</tr>
<% while NOT OraDynaset.EOF %>
<tr>
<% For col = 0 to Columns.Count - 1 %>
<td><% = Columns(col) %>
</td>
```

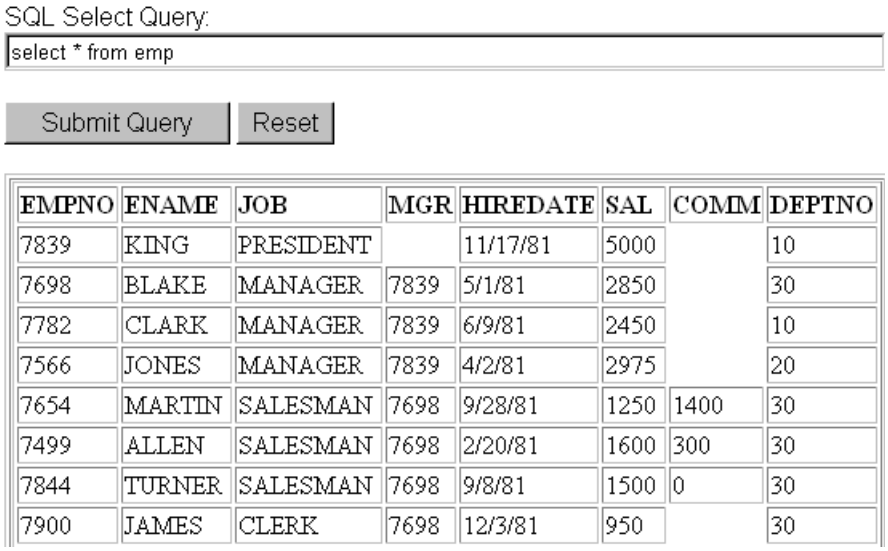
```
<% Next %>
</tr>
<% OraDynaset.MoveNext %>
<% WEnd %>
</table>

<p></font><%End If%> </p>
<hr>
</td>
</tr>
</table>
</body>
</html>
```

3. Create a virtual directory from Microsoft Internet Service Manager with read and execute access, and place all `.asp` and `.asa` files in that directory.
4. Create an HTML page from which to launch the `oo4odemo.asp` file. Add a link in the page as follows:

   ```
   <a href="/<your_path>/OO4ODEMO.ASP">This link launches the demo!</a>
   ```

5. Load the page in a web browser and click the link to the demonstration.
6. Enter a query, such as `'SELECT * FROM EMP'`, in the **SQL SELECT Query** field, and select the **Submit Query** button. Do not include a semicolon (`;`) at the end of the query.



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/81	5000		10
7698	BLAKE	MANAGER	7839	5/1/81	2850		30
7782	CLARK	MANAGER	7839	6/9/81	2450		10
7566	JONES	MANAGER	7839	4/2/81	2975		20
7654	MARTIN	SALESMAN	7698	9/28/81	1250	1400	30
7499	ALLEN	SALESMAN	7698	2/20/81	1600	300	30
7844	TURNER	SALESMAN	7698	9/8/81	1500	0	30
7900	JAMES	CLERK	7698	12/3/81	950		30

****

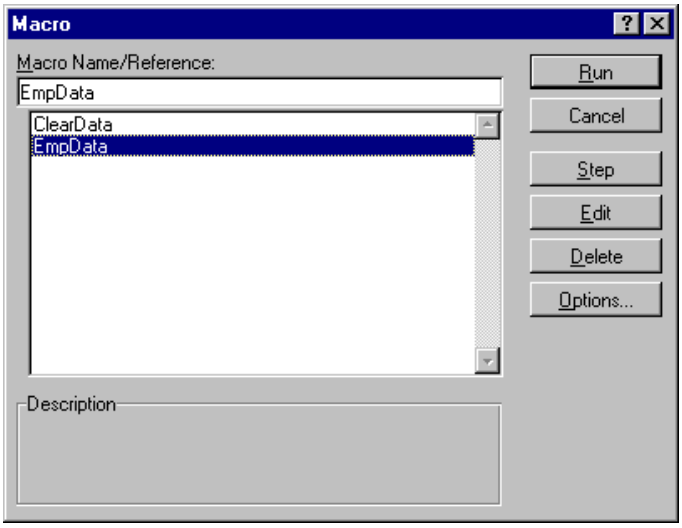
Using Oracle Objects for OLE Automation with Excel

This sample shows how to insert Oracle data into an Excel worksheet.

> **Note:** The sample code for this example is available in the *ORACLE_BASE**ORACLE_HOME*\oo4o\excel\samples\ directory.

To use OLE Automation with Microsoft Excel to insert Oracle data into a worksheet, perform the following steps:

1. Start Excel and create a new worksheet.
2. Use the **Macro** options in the **Tools** menu to create and edit new macros for manipulating the Oracle data.



3. Enter Visual Basic code for macros to create and access an Oracle dynaset, such as the following `EmpData()` and `ClearData()` procedures (macros):

```
Sub EmpData()

'Declare variables
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim EmpDynaset As OraDynaset
Dim flds() As OraField
Dim fldcount As Integer
  Set OraSession = CreateObject("OracleInProcServer.XOraSession")
  Set OraDatabase = OraSession.OpenDatabase("ExampleDB", "scott/tiger", 0&)
  Set EmpDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)
  Range("A1:H15").Select
  Selection.ClearContents

   'Declare and create an object for each column.
   'This will reduce objects references and speed up your application.
fldcount = EmpDynaset.Fields.Count
ReDim flds(0 To fldcount - 1)
For Colnum = 0 To fldcount - 1
  Set flds(Colnum) = EmpDynaset.Fields(Colnum)
Next

'Insert Column Headings
For Colnum = 0 To EmpDynaset.Fields.Count - 1
  ActiveSheet.Cells(1, Colnum + 1) = flds(Colnum).Name
Next

'Display Data
For Rownum = 2 To EmpDynaset.RecordCount + 1
  For Colnum = 0 To fldcount - 1
    ActiveSheet.Cells(Rownum, Colnum + 1) = flds(Colnum).Value
```

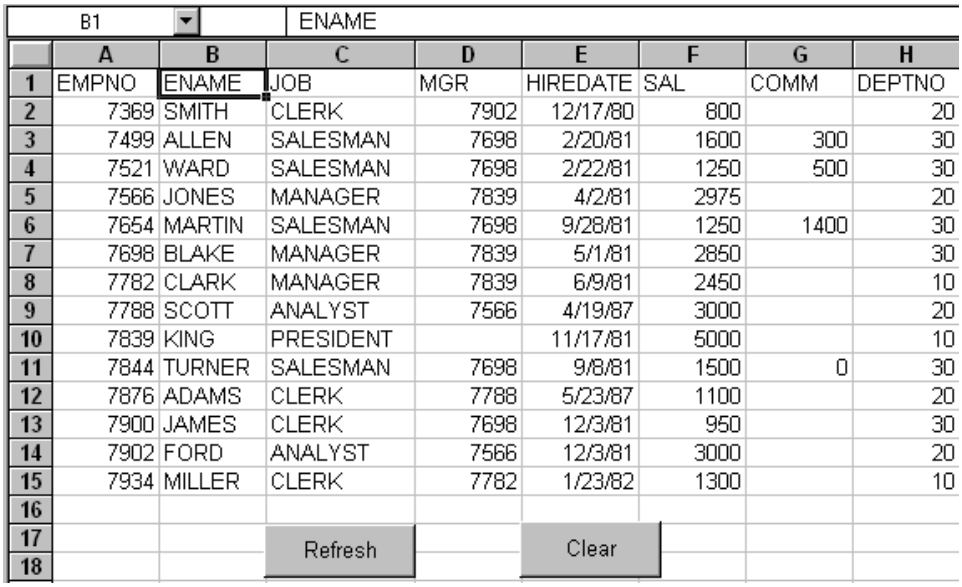
```
  Next
  EmpDynaset.MoveNext
Next

Range("A1:A1").Select

End Sub

Sub ClearData()
  Range("A1:H15").Select
  Selection.ClearContents
  Range("A1:A1").Select
End Sub
```

4. Assign the procedures (macros) that were created, such as `EmpData()` and `ClearData()`, to command buttons in the worksheet for easy access. When you select the buttons, you can clear and refresh the data in the worksheet. In the following screenshot, `ClearData()` is assigned to the Clear button and `EmpData()` is assigned to the Refresh button.



	A	B	C	D	E	F	G	H
1	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
2	7369	SMITH	CLERK	7902	12/17/80	800		20
3	7499	ALLEN	SALESMAN	7698	2/20/81	1600	300	30
4	7521	WARD	SALESMAN	7698	2/22/81	1250	500	30
5	7566	JONES	MANAGER	7839	4/2/81	2975		20
6	7654	MARTIN	SALESMAN	7698	9/28/81	1250	1400	30
7	7698	BLAKE	MANAGER	7839	5/1/81	2850		30
8	7782	CLARK	MANAGER	7839	6/9/81	2450		10
9	7788	SCOTT	ANALYST	7566	4/19/87	3000		20
10	7839	KING	PRESIDENT		11/17/81	5000		10
11	7844	TURNER	SALESMAN	7698	9/8/81	1500	0	30
12	7876	ADAMS	CLERK	7788	5/23/87	1100		20
13	7900	JAMES	CLERK	7698	12/3/81	950		30
14	7902	FORD	ANALYST	7566	12/3/81	3000		20
15	7934	MILLER	CLERK	7782	1/23/82	1300		10
16								
17								
18								

Using Microsoft C++

For details about Oracle Objects for OLE with Visual C++, see *Oracle Objects for OLE C++ Class Library Developer's Guide*, available as online help.

Using Oracle Data Control with Visual Basic

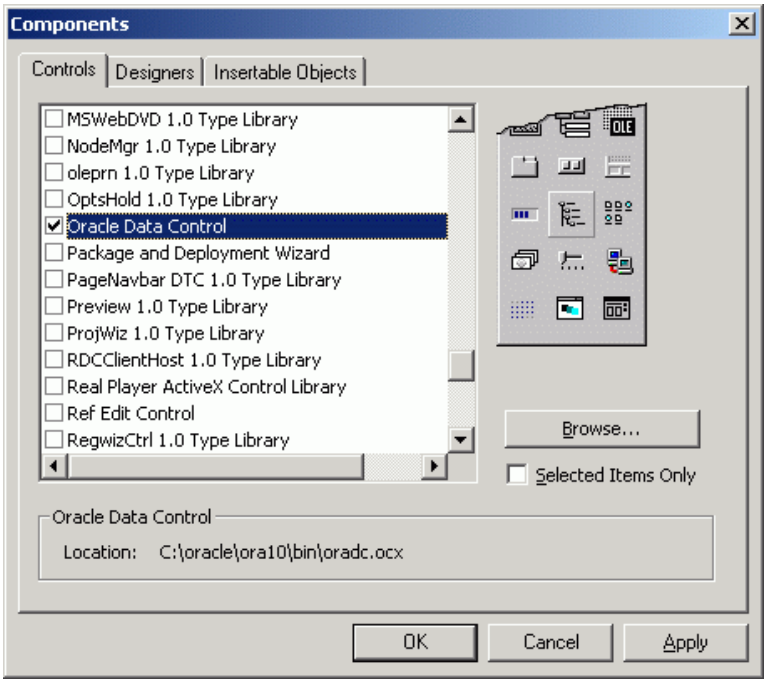
Oracle Data Control, when refreshed, automatically creates a client (if needed), session, database, and dynaset. For a basic application, little or no code is required.

This section shows two ways to set the properties of Oracle Data Control:

- Using the Visual Basic Properties window
- Programming the properties

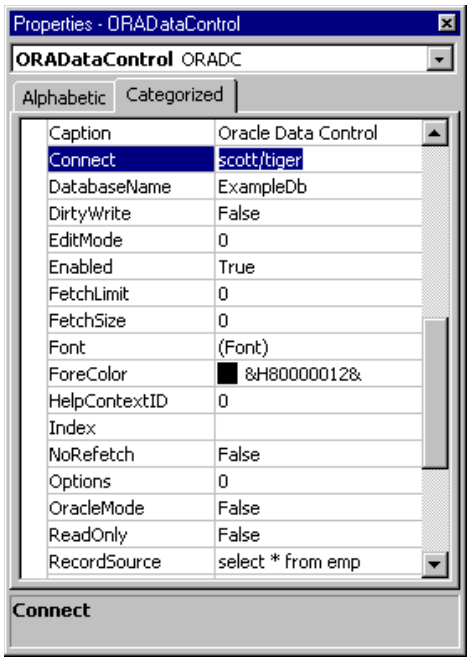
Setting Oracle Data Control Properties with the Properties Window

1. Start Visual Basic and create a new project.
2. In the **Components** option of the **Project** menu, add **Oracle Data Control** to the project.

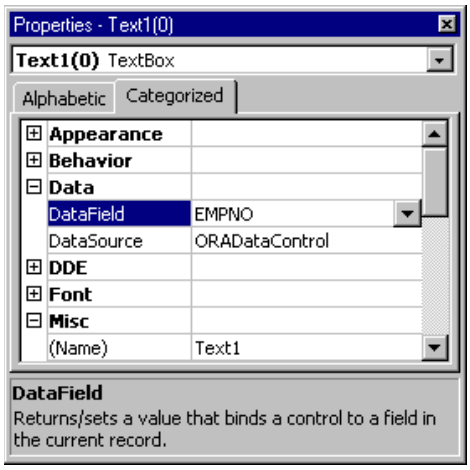


The Oracle Data Control is added to your Visual Basic tool palette and looks like this:

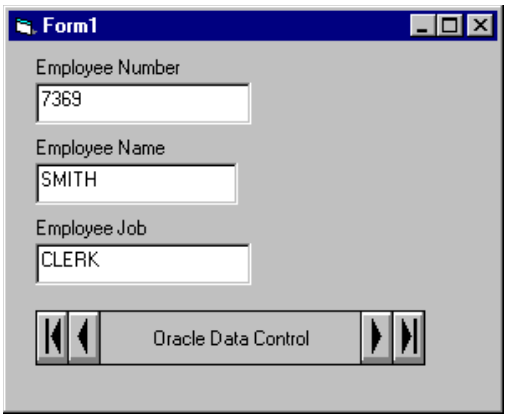
3. To add Oracle Data Control to a project, drag and drop the control onto a form. Resize and position the control.
4. Change the name of the control to `OraDataControl`. Set up the `Connect`, `DatabaseName`, and `RecordSource` properties as follows to access the Oracle database:



5. When Oracle Data Control is set up, you can drag and drop a Visual Basic control onto the same form and access the data in the control. Set the **Data** properties to access the data field and source that you want. The following figure shows a TextBox control that sets up display of the employee numbers.



6. When the project is run, the data identified by the `RecordSource` property is displayed using Oracle Data Control.



You can also use Microsoft Grid Control to display all the data in the table. You need to add the grid control with the Components option of the Project menu.

Setting Oracle Data Control Properties Programmatically

The following code fragment demonstrates how to programmatically set the properties of **Oracle Data Control** required to create a dynaset. These are the same properties that you can set with the **Properties** window in Visual Basic.

1. Create a new project, and then in the **Components** option of the **Project** menu, add Oracle Data Control to the project.
2. Drag and drop **Oracle Data Control** onto a form. Change the name of the control to `OraDataControl`.
3. After you have inserted **Oracle Data Control** onto a form, add the following code to the load procedure associated with the form:

```
...

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

'Set the database name.
OraDataControl.DatabaseName = "ExampleDb"

'Set the record source.
OraDataControl.RecordSource = "select * from emp"

'Refresh the data control.
OraDataControl.Refresh
...
```

You now have a valid session, database, and dynaset that can be referenced as follows:

Object	Reference
`orasession`	`oradatacontrol.oradatabase.orasession`
`oradatabase`	`oradatacontrol.oradatabase`
oradynaset	`oradatacontrol.recordset`

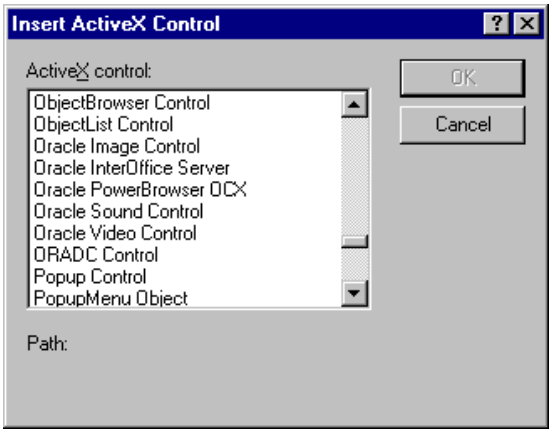
4. You can access the data in the `RecordSource` property using Visual Basic controls, such as the **TextBox**, as shown in the previous example.

Using the Oracle Data Control with MS Visual C++

This example shows how to create a basic Win32 Application with **Oracle Data Control** using MS Visual C++. This example assumes that both the Oracle data and DB Grid controls were registered on the system.

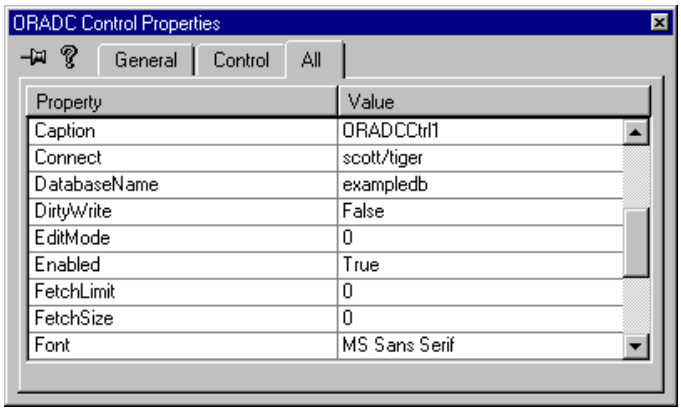
1. Start the Microsoft Visual C++ program.
2. From the **File** Menu, select **New**.
3. In the **Projects** tab of the **New Window**, select **MFC AppWizard.exe**. Enter a project name, such as OO4O, and determine the location of the project. Click **OK**.
4. In Step 1 of the **MFC AppWizard**, select **Dialog based application**, then click **Next**.
5. In Step 2 of the wizard, make sure the **ActiveX Controls** box is checked; accept the defaults; and enter a title for the dialog box. Click **Next**.
6. In Step 3 of the wizard, accept the defaults. Click **Next**.
7. In Step 4, click **Finish**. At the **New Project Information** screen, click **OK**.
8. In the **Project Workspace** dialog box, select the **ResourceView** tab. Expand the **Resources** folder, then expand the **Dialog** folder.
9. Double-click the main project dialog box to edit the dialog box.

   Note: If you used OO4O as the project name, it is named `IDD_OO4O_DIALOG`.
10. Delete the default controls that are on the dialog box. Resize the dialog box to make it larger.
11. With the dialog box selected, click the right mouse button to display the menu. Select **Properties** from the menu. In the **General** tab of the **Properties** window, change the caption to **Oracle Data Control Example**. Close the **Properties** window.
12. With the dialog box selected, click the right mouse button to display the menu. Select **Insert ActiveX Control...** from the menu. Select **ORADC Control** in the window and then click **OK**.

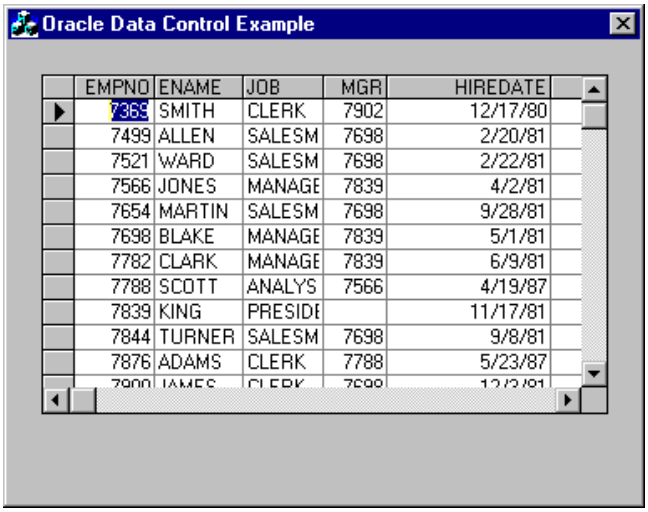


13. Position the **ORADC Control** at the bottom of the dialog box. With the data control selected, click the right mouse button to display the menu. Select **Properties** from the menu.
14. In the **General** tab of the **Properties window**, change the `ID` to `IDC_ORADATACONTROL`. Deselect the check mark for **Visible** so that the control is hidden when the application is run.
15. Display the **All** tab of the **Properties window** and set the following:

```
Connect: scott/tiger
DatabaseName: exampledb
RecordSource: select * from emp
```



16. With the dialog box selected, click the right mouse button to display the menu. Select the **Insert ActiveX** control from the menu. Locate the **DBGrid Control** and click **OK**.
17. Position the **DBGrid Control** at the top of the dialog box and resize it. Display the properties for the control. In the **All** tab of the **Properties** window, set the `DataSource` property to **Oracle Data Control** (`IDC_ORADATACONTROL`). Accept the defaults for the other properties. These can be changed later.
18. From the **File** Menu, select **Save All**.
19. Build and Execute the project.
20. The **DBGrid Control** displays the records from the `emp` table as in the following illustration:


Oracle Data Control Example
EMPNO ENAME JOB MGR HIREDATE
7369 SMITH CLERK 7902 12/17/80
7499 ALLEN SALESM 7698 2/20/81
7521 WARD SALESM 7698 2/22/81
7566 JONES MANAGE 7839 4/2/81
7654 MARTIN SALESM 7698 9/28/81
7698 BLAKE MANAGE 7839 5/1/81
7782 CLARK MANAGE 7839 6/9/81
7788 SCOTT ANALYS 7566 4/19/87
7839 KING PRESIDE 11/17/81
7844 TURNER SALESM 7698 9/8/81
7876 ADAMS CLERK 7788 5/23/87

3

Basic Features

This chapter describes basic features of Oracle Objects for OLE.

This chapter contains these topics:

Overview of Client Applications

Oracle Objects for OLE enables client applications to connect to Oracle databases, execute commands, and access and manipulate the results returned. While some flexibility exists in the order in which specific tasks can be performed, every application using OO4O Automation objects performs the following basic steps:

- Accessing the Oracle Objects for OLE Automation Server
- Connecting to Oracle Database
- Executing Commands
- Disconnect from the servers and free the OO4O objects used

Accessing the Oracle Objects for OLE Automation Server

To connect to an Oracle database with the OO4O Automation Server, you must first create an instance of the server. In Visual Basic (VB), this is usually done by calling the `CreateObject` method, although the `NEW` keyword can also be used.

You can use the Visual Basic `CreateObject` method with either of the following two OO4O server objects. The interfaces of these objects can provide access to OO4O and enable a connection to Oracle Database.

- `OraSession`

  Highest level object for an application. It manages collections of `OraDatabase`, `OraConnection`, and `OraDynaset` objects.

- `OraServer`

  Represents a physical connection to a database instance and allows for connection multiplexing

The `CreateObject` method uses the `ID` of the component and object as arguments.

Obtaining an OraSession Object

The following script demonstrates how to obtain an `OraSession` object in Visual Basic. `OO4OSession` is the object variable that holds an instance of the `OraSession` object.

```
Dim OO4OSession as Object
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
```

or

```
Dim OO4OSession as OraSession
Set OO4OSession = New OraSessionClass
```

or

```
Dim OO4OSession as New OraSessionClass
```

The following example demonstrates how to obtain an `OraSession` object in IIS Active Server Pages.

```
<OBJECT RUNAT=Server SCOPE=APPLICATION ID=OO4OSession
                PROGID="OracleInProcServer.XOraSession">
</OBJECT>
```

`OracleInProcServer.XOraSession` is the version independent program `ID` for OO4O that the Oracle client installation program registers in the Windows registry. It is the symbolic name for a globally unique identifier (`CLSID`) that identifies the OO4O component.

Obtaining an OraServer Object

You can also use the `OraServer` object interface for accessing the OO4O Automation Server.

```
Dim OO4OServer as Object
Set OO4OServer = CreateObject("OracleInProcServer.XOraServer")
```

Now you can connect to Oracle Database.

See Also: "Connecting to Oracle Database" on page 3-2

Connecting to Oracle Database

Once you have obtained an interface, you can use it to establish a user session in an Oracle database by invoking the `OpenDatabase` method.

```
Set EmpDb= OO4OSession.OpenDatabase("ExampleDb", "Scott/Tiger", 0)
```

or

```
Set EmpDb= OO4OServer.OpenDatabase("Scott/Tiger")
```

The variable `EmpDb` represents a user session. It holds an `OraDatabase` interface and can be used to send commands to Oracle Database using *`ExampleDb`* for the network connection alias and `scott/tiger` for the user name and password.

See Also: OpenDatabase Method on page 10-212

Using OraServer for Connection Multiplexing

The `OraServer` interface allows multiple user sessions to share a physical network connection to the database. This reduces resource usage on the network and the database, and allows for better server scalability. However, execution of commands by multiple user sessions is serialized on the connection. Therefore, this feature is not recommended for use in multithreaded applications in which parallel command execution is needed for performance.

The following code example shows how to use the `OraServer` interface to establish two user sessions:

```
Set OO4OServer = CreateObject("OracleInProcServer.XOraServer")
OO4OServer.Open("ExampleDb")
Set EmpDb1 = OO4OServer.OpenDatabase("Scott/Tiger")
Set EmpDb2 = OO4OServer.OpenDatabase("Scott/Tiger")
```

You can also obtain user sessions from a previously created pool of objects.

See Also: "Using the Connection Pool Management Facility" on page 3-8

Executing Commands

Commands that can be sent to Oracle databases using OO4O Automation objects are divided into the following categories:

- "Queries" on page 3-3
- "Data Manipulation Language Statements" on page 3-5

Queries

Queries are statements that retrieve data from a database. A query can return zero, one, or many rows of data. All queries begin with the SQL keyword `SELECT`, as in the following example:

```
SELECT ename, empno FROM emp
```

In OO4O, `SELECT` statements such as this are used with the `CreateDynaset` method of the `OraDatabase` interface to execute queries. This method returns an `OraDynaset` object that is then used to access and manipulate the set of rows returned. An `OraDynaset` object encapsulates the functions of a client-side scrollable (forward and backward) cursor that allows browsing the set of rows returned by the query it executes.

Note: Caching result sets on the client's local disk can be disabled if backward scrollability is not a requirement. This is strongly recommended and can provide significant performance improvements. Passing the ORADYN_NOCACHE option in the CreateDynaset method disables caching. This constant is defined in the oraconst.txt file and can be found in the root directory where OO4O is installed, *ORACLE_BASE**ORACLE_HOME*\OO4O.

See Also:

- OraDynaset Object on page 9-30
- CreateDynaset Method on page 10-85

The following code example shows how to connect to the *ExampleDb* database, execute a query, move through the result set of rows, and displays the column values of each row in a simple message box.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "Scott/Tiger", 0 )

' SELECT query described above used in next line
Set Employees = EmpDb.CreateDynaset("SELECT ename, empno FROM" & _
          "emp",ORADYN_NOCACHE)
While NOT Employees.EOF
  MsgBox "Name: " & Employees("ENAME").value & "Employee #: " & _
          Employees("EMPNO").value
Employees.MoveNext
Wend
```

In the previous example, Employees("ENAME") and Employees("EMPNO") return values of the ENAME and the EMPNO columns from the current row in the result set, respectively. An alternative method of accessing the column values is to use the positions of the columns, Employees(0) for the ENAME column and Employee(1) for EMPNO. This method obtains the column value faster than referencing a column by its name.

The Employees.MoveNext statement in the example sets the current row of the result set to the next row. The EOF property of the OraDynaset is set to True if an attempt is made to move past the last row in the result set.

The MoveNext method is one navigational method in the OraDynaset interface. Other methods include MoveFirst, MoveLast, MoveNext, MovePrevious, MoveNextn, MovePreviousn, MoveRel, and MoveTo.

An OraDynaset object also provides methods to update and delete rows retrieved from base tables or views that can be updated. In addition, it provides a way to insert new rows. See "OraDynaset Object" on page 9-30.

Queries can also require the program to supply data to the database using input (bind) variables, as in the following example:

```
SELECT name, empno
        FROM employees
        WHERE ename = :ENAME
```

In the SQL statement, :ENAME is a placeholder for a value that is supplied by the application.

In OO4O, the OraParameter object is used to supply data values for placeholders.

To define a parameter, use the OraParameters collection object. This object is obtained by referencing the Parameters property of an OraDatabase interface. The OraParameters collection provides methods for adding, removing, and obtaining references to OraParameter objects.

The following statement adds an input parameter, ORAPARM_INPUT, to the OraParameters collection contained in the EmpDb object.

```
EmpDb.Parameters.Add "ENAME", "JONES", ORAPARM_INPUT
```

ENAME is the name of the parameter and must be the same as the name of the placeholder in the SQL statement, :ENAME in the sample code. JONES is provided as the initial value, and ORAPARM_INPUT notifies OO4O that it is used as an INPUT parameter.

The following example creates an OraDynaset object that contains only one row for an employee whose name is 'JONES'.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "Scott/Tiger", 0 )
EmpDb.Parameters.Add "ENAME", "JONES", ORAPARM_INPUT
Set Employees = EmpDb.CreateDynaset("SELECT ename, empno FROM emp" & _
           "WHERE ename = :ENAME",ORADYN_NOCACHE)

While NOT Employees.EOF
   MsgBox "Name: " & Employees("ename").value & "Employee #: " & _
            Employees("empno").value
   Employees.MoveNext
Wend
```

See Also:

- OraParameter Object on page 9-50
- OraParameters Collection on page 9-68
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- MovePreviousn, MoveNextn, MoveRel, and MoveTo Methods on page 10-202
- OraDatabase Object on page 9-28

Data Manipulation Language Statements

Data manipulation language (DML) statements can change data in the database tables. For example, DML statements are used to:

- Updating Database Records
- Deleting Rows from a Table
- Inserting New Rows into a Table

The OraDatabase interface in OO4O provides two methods for executing DML statements: ExecuteSQL and CreateSQL. The following discussion describes how these methods can be used to execute various types of DML statements.

See Also:

- ExecuteSQL Method on page 10-144
- CreateSQL Method on page 10-111

Updating Database Records

The following example uses the `ExecuteSQL` method to execute an `UPDATE` statement.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
EmpDb.Parameters.Add "ENAME", "JONES", ORAPARM_INPUT
EmpDb.ExecuteSQL ("UPDATE emp SET sal = sal + 1000 WHERE ename = :ENAME")
```

Another way to execute the `UPDATE` statement is to use the `CreateSQL` method:

```
Set sqlStatement = EmpDb.CreateSQL("UPDATE emp SET sal = sal + 1000" & _
             "WHERE ename = :ENAME", 0&)
```

Both the `ExecuteSQL` and `CreateSQL` methods execute the UPDATE statement provided. The difference is that the `CreateSQL` method returns a reference to an `OraSQLStmt` interface, in addition to executing the statement. This interface can later be used to execute the same query using the `Refresh` method. Because the query has already been parsed by the database, subsequent execution of the same query results in faster execution, especially if bind parameters are used.

For example, to increase the salary of an employee named `KING` by 1000, change the value of the placeholder, and refresh the `sqlStatement` object as follows:

```
EmpDb.Parameters("ENAME").Value = "KING"
sqlStatement.Refresh
```

For DML statements that are frequently executed, using parameters with `OraSqlStmt` objects is more efficient than using the `ExecuteSql` statement repeatedly. When the `Refresh` method of the `OraSQLStmt` is executed, the statement no longer needs to be parsed by the database. In application servers, such as Web servers, where the same queries are frequently executed with different parameter values, this can lead to significant savings in Oracle Database processing.

See Also:

- ExecuteSQL Method on page 10-144
- CreateSQL Method on page 10-111
- OraSQLStmt Object on page 9-60

Deleting Rows from a Table

The following example uses the `CreateSQL` method to delete rows from the `emp` table.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
EmpDb.Parameters.Add "ENAME", "JONES", ORAPARM_INPUT
Set sqlStatement = EmpDb.CreateSQL ("DELETE from emp WHERE ename = :ENAME")
```

To delete another row from the `emp` table, the value of the parameter is changed, and the `sqlStatement` object is refreshed.

```
EmpDb.Parameters("ENAME").Value = "KING"
```

```
sqlStatement.Refresh
```

Inserting New Rows into a Table

The following example adds a new row into the table.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
EmpDb.ExecuteSQL ("INSERT INTO emp (empno, ename, job, mgr, deptno)" & _
          "VALUES (1233,'OERTEL', 'WRITER', 7839, 30) ")
```

Inserting Multiple Rows Using Parameter Arrays

You can use parameter arrays to fetch, update, insert, or delete multiple rows in a table. Using parameter arrays for manipulating multiple rows is more efficient than executing multiple statements that operate on individual rows.

The following example demonstrates how the `AddTable` method of the `OraDatabase` interface is used to create parameter arrays. The arrays are then populated with values, and used as placeholders in the execution of an `INSERT` statement that inserts two rows into the `emp` table.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("Exampledb", "scott/tiger", 0)

'Creates parameter arrays for the empno, ename, job, and salary columns
EmpDb.Parameters.AddTable "EMPNO_ARRAY", ORAPARM_INPUT, ORATYPE_NUMBER, 2
EmpDb.Parameters.AddTable "ENAME_ARRAY", ORAPARM_INPUT, ORATYPE_VARCHAR2, 2, 10
EmpDb.Parameters.AddTable "JOB_ARRAY", ORAPARM_INPUT, ORATYPE_VARCHAR2, 2, 9
EmpDb.Parameters.AddTable "MGR_ARRAY", ORAPARM_INPUT, ORATYPE_NUMBER, 2
EmpDb.Parameters.AddTable "DEPT_ARRAY", ORAPARM_INPUT, ORATYPE_VARCHAR2, 2, 10
Set EmpnoArray = EmpDb.Parameters("EMPNO_ARRAY")
Set EnameArray = EmpDb.Parameters("ENAME_ARRAY")
Set JobArray = EmpDb.Parameters("JOB_ARRAY")
Set MgrArray = EmpDb.Parameters("MGR_ARRAY")
Set DeptArray = EmpDb.Parameters("DEPT_ARRAY")

'Populate the arrays with values
EmpnoArray(0) = 1234
EnameArray(0) = "JORDAN"
JobArray(0) = "SALESMAN"
MgrArray(0) = 7839
DeptArray(0) = 30
EmpnoArray(1) = 1235
EnameArray(1) = "YOUNG"
JobArray(1) = "SALESMAN"
MgrArray(1) = 7839
DeptArray(1) = 30

'Insert two rows
EmpDb.ExecuteSQL ("INSERT INTO emp (empno, ename, job, mgr, deptno) VALUES" & _
          "(:EMPNO_ARRAY,:ENAME_ARRAY, :JOB_ARRAY,:MGR_ARRAY, :DEPT_ARRAY)")
```

See Also: AddTable Method on page 10-23

Thread Safety

OO4O is thread-safe and can be used effectively in multithreaded applications and environments such as the Microsoft Internet Information Server (IIS). OO4O supports both the free and apartment threading models in COM/DCOM.

Access to OO4O object attributes is serialized when used with multiple threads of execution. To achieve maximum concurrency in query execution in a multithreaded application with OO4O, avoid sharing objects in multiple threads.

Avoid using commit and rollback operations on a session object that is shared among multiple threads because all connections associated with that session are committed or rolled back. To perform commit and rollback operations on a session object, create a unique session object for each database object used.

Using the Connection Pool Management Facility

The connection pool in OO4O is a pool of `OraDatabase` objects. An OO4O connection pool is a group of (possibly) already connected `OraDatabase` objects. For applications that require constant connections and disconnections to the database, such as ASP Web applications, using a connection pool results in enhanced performance.

Creating the Connection Pool

The connection pool is created by invoking the `CreateDatabasePool` method of the `OraSession` interface. An `OraDatabase` object represents a connection to an Oracle database and contains methods for executing SQL statements and PL/SQL blocks.

See Also: CreateDatabasePool Method on page 10-83

Obtaining from and Returning Objects to the Pool

To retrieve an `OraDatabase` object from the pool, call the `GetDatabaseFromPool` method. This function returns a reference to an `OraDatabase` object.

See Also: GetDatabaseFromPool Method on page 10-155

Destroying the Pool

The pool is implicitly destroyed if the parent session object that it belongs to is destroyed. It can also be destroyed at any time by invoking the `DestroyDatabasePool` method.

See Also: DestroyDatabasePool Method on page 10-128

Accessing the Pool attributes

The following are the database pool properties. These properties are read-only:

- `DbPoolMaxSize` - maximum pool size
- `DbPoolCurrentSize` - current size of the pool
- `DbPoolInitialSize` - initial size of the pool

Processing Transactions Using the Database from the Connection Pool

The following example shows the recommended way to process transactions:

```
set Odb = OraSession.GetDatabaseFromPool(0)
Odb.Connection.BeginTrans
…

Odb.Connection.CommitTrans
```

Detection of Lost Connections

OO4O, linked with clients from releases 8.1.6 or higher, supports detection of lost connections.

Applications can verify the status of the database connection by invoking the `ConnectionOK` property of the `OraDatabase` object. The `OraSession.GetDatabaseFromPool` method now verifies the connection before returning the `OraDatabase` to the application.

If the connection is lost, the `GetDatabaseFromPool` method drops the lost connection and fetches a new connection.

```
Dim MyDatabase As OraDatabase
Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set MyDatabase = MySession.OpenDatabase("ora90", "scott/tiger", 0&)

' Other code
...
' Check if the database connection has not timed out
if MyDatabase.ConnectionOK
  MsgBox " The database connection is valid"
endif
```

See Also:

- ConnectionOK Property on page 11-26
- OraDatabase Object on page 9-28
- GetDatabaseFromPool Method on page 10-155

PL/SQL Support

PL/SQL is the Oracle procedural extension to the SQL language. PL/SQL processes complicated tasks that simple queries and SQL data manipulation language statements cannot perform. Without PL/SQL, Oracle Database would have to process SQL statements one at a time. Each SQL statement results in another call to the database and consequently higher performance overhead. In a networked environment, the overhead can be significant. Every time a SQL statement is issued, it must be sent over the network, creating more traffic. However, with PL/SQL, an entire block of statements can be sent to a database at one time. This can greatly reduce communication between an application and a database.

PL/SQL allows a number of constructs to be grouped into a single block and executed as a unit. These include:

- One or more SQL statements
- Variable declarations
- Assignment statements
- Procedural control statements (`IF...THEN...ELSE` statements and loops)
- Exception handling statements
- Calls to other Oracle stored procedures and stored functions
- Special PL/SQL features such as records, tables, and cursor `FOR` loops
- Cursor variables

PL/SQL Integration with Oracle Objects for OLE

Oracle Objects for OLE (OO4O) provides tight integration with PL/SQL stored procedures. OO4O supports PL/SQL stored procedures, PL/SQL tables, PL/SQL, cursors and so on. The PL/SQL bind variables are supported through the `OraParameter Add` method.

The stored procedure block is executed either through the `CreateSQL` method or the `ExecuteSQL` method.

Oracle Objects for OLE can return a cursor created in the stored procedure or anonymous PL/SQL block as a `READONLY` dynaset object.To do this, you must assign the cursor variable as an `OraParameter` object of type `ORATYPE_CURSOR`.

After executing the stored procedure, the `Value` property of this `OraParameter` object returns a read-only dynaset object.

This dynaset object can be treated the same as other dynaset objects.

See Also:

- AddTable Method on page 10-23
- Add Method on page 10-8
- ExecuteSQL Method on page 10-144
- CreateSQL Method on page 10-111
- Value Property on page 11-173
- OraParameter Object on page 9-50

Executing PL/SQL Blocks Using ExecuteSQL and CreateSQL

In OO4O, you can use the `ExecuteSQL` or `CreateSQL` methods of the `OraDatabase` object to execute PL/SQL blocks, as the following example shows:

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)

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

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

'Add SAL as an Output parameter
EmpDb.Parameters.Add "SAL", 0, ORAPARM_OUTPUT
EmpDb.Parameters("SAL").ServerType = ORATYPE_NUMBER

'Add COMMISSION as an Output parameter and set its initial value.
EmpDb.Parameters.Add "COMMISSION", 0, ORAPARM_OUTPUT
EmpDb.Parameters("COMMISSION").ServerType = ORATYPE_NUMBER
EmpDb.ExecuteSQL ("BEGIN SELECT ename, sal, comm INTO :ENAME, :SAL," & _
             ":COMMISSION FROM emp WHERE empno = :EMPNO; END;")

'display the values of Ename, Sal, Commission parameters
MsgBox "Name: " & EmpDb.Parameters("ENAME").Value
MsgBox "Salary " & EmpDb.Parameters("SAL").Value
MsgBox "Commission: " & EmpDb.Parameters("COMMISSION").Value
```

The following example executes a PL/SQL block that calls a stored procedure using the `CreateSQL` method in OO4O. The procedure takes a department number as input and returns the name and location of the department.

This example is used for creating the stored procedure in the employee database.

```
CREATE OR REPLACE PACKAGE Department as
PROCEDURE GetDeptName (inDeptNo IN NUMBER, outDeptName OUT VARCHAR2,
                   outDeptLoc OUT VARCHAR2);
END Department;
/

CREATE OR REPLACE PACKAGE BODY Department as
PROCEDURE GetDeptName(inDeptNo IN NUMBER, outDeptName OUT VARCHAR2,
                 outDeptLoc OUT VARCHAR2) is
BEGIN
   SELECT dname, loc into outDeptName, outDeptLoc from DEPT
      WHERE deptno = inDeptNo;
   END;
END Department;
/
```

The following example executes the previously created procedure to get the name and location of the department where `deptno` is `10`.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
empDb.Parameters.Add "DEPTNO", 10, ORAPARM_INPUT
empDb.Parameters("DEPTNO").ServerType = ORATYPE_NUMBER
empDb.Parameters.Add "DNAME", 0, ORAPARM_OUTPUT
empDb.Parameters("DNAME").ServerType = ORATYPE_VARCHAR2
empDb.Parameters.Add "DLOC", 0, ORAPARM_OUTPUT
empDb.Parameters("DLOC").ServerType = ORATYPE_VARCHAR2
Set PlSqlStmt = empDb.CreateSQL("Begin Department.GetDeptname" & _
             "(:DEPTNO, :DNAME, :DLOC); end;", 0&)

'Display Department name and location
MsgBox empDb.Parameters("DNAME").Value & empDb.Parameters("DLOC").Value
```

See Also:

- ExecuteSQL Method on page 10-144
- CreateSQL Method on page 10-111

Returning PL/SQL Cursor Variables

PL/SQL cursor variables are mainly used for accessing one or more query result sets from PL/SQL blocks and stored procedures and functions. The `OraParameter` object in OO4O can be used to hold a PL/SQL cursor variable.

The `OraParameter` object representing a cursor variable should be of type `ORATYPE_CURSOR`, and can only be defined as an output variable. After the PL/SQL block is executed, the `Value` property of the `OraParameter` object contains a read-only `OraDynaset` object. This `OraDynaset` object can be used to scroll through the returned rows.

In some cases, it is better to use the `CreateSQL` method for executing PL/SQL procedures than the `ExecuteSQL` method. The `Refresh` method on the `OraSQLStmt` object can result in modified PL/SQL cursors. If the `CreateSQL` method is used, these

modified cursors are automatically associated with the existing dynaset object, and no new dynaset object is created.

> **See Also:** "Executing PL/SQL Blocks Using ExecuteSQL and CreateSQL" on page 3-10

You cannot set the SQL property of the dynaset object; this raises an error.

> **Note:** PL/SQL stored procedures that contain cursors as table parameters are not supported.

You should call the `Remove` method on the parameter object. This helps in cleaning the dynaset object and local temporary cache files.

The following example contains a stored procedure that gets the cursors for the `emp` and `dept` tables and a small application that executes the procedure.

Stored Procedure

```
CREATE PACKAGE EmpAndDept AS
   cursor emp is select * from emp;
   cursor dept is select * from dept;
   TYPE EmpCurTyp IS REF CURSOR RETURN emp%ROWTYPE;
   TYPE DeptCurTyp IS REF CURSOR RETURN dept%ROWTYPE;
   PROCEDURE GetEmpAndDeptData (emp_cv OUT EmpCurTyp,
                               dept_cv OUT DeptCurTyp);
END EmpAndDept;
/


CREATE PACKAGE BODY EmpAndDept AS
   PROCEDURE GetEmpAndDeptData (emp_cv OUT EmpCurTyp,
                                dept_cv OUT DeptCurTyp) IS
       BEGIN
         OPEN emp_cv FOR SELECT * FROM emp;
         OPEN dept_cv FOR SELECT * FROM dept; END GetEmpAndDeptData;
END EmpAndDept;
/
```

Application

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
empDb.Parameters.Add "EMPCUR", 0, ORAPARM_OUTPUT
empDb.Parameters("EMPCUR").serverType = ORATYPE_CURSOR
empDb.Parameters.Add "DEPTCUR", 0, ORAPARM_OUTPUT
empDb.Parameters("DEPTCUR").serverType = ORATYPE_CURSOR
Set PlSqlStmt = empDb.CreateSql("Begin EmpAndDept.GetEmpAndDeptData (:EMPCUR," & _
             ":DEPTCUR); end;", 0)
Set EmpDynaset = empDb.Parameters("EmpCur").Value
Set DeptDynaset = empDb.Parameters("DeptCur").Value
MsgBox EmpDynaset.Fields("ENAME").Value
MsgBox DeptDynaset.Fields("DNAME").Value
```

See Also:

- OraParameter Object on page 9-50
- ServerType Property on page 11-138
- Value Property on page 11-173
- OraDynaset Object on page 9-30
- OraSQLStmt Object on page 9-60
- DynasetOption Property on page 11-50
- CreateSQL Method on page 10-111
- Refresh Method on page 10-225
- Remove Method on page 10-230
- DynasetCacheParams Method on page 10-133
- Recordset Property on page 14-29

Returning PL/SQL Tables

PL/SQL tables are mainly used for accessing arrays of PL/SQL data. The `OraParamArray` object in OO4O can be used to hold a PL/SQL cursor variable.

The `OraParamArray` object representing a table variable should be created first the using the `AddTable` method. Table values are accessed or set using the `Get_Value` and `Put_Value` methods of the `OraParamArray` object.

The PL/SQL procedure `GetEmpNamesInArray` returns an array of `ENAME` values for array of `EMPNO`s.

```
CREATE PACKAGE EmpNames AS
    type NUMARRAY is table of NUMBER index by
          BINARY_INTEGER; --Define EMPNOS array
    type VCHAR2ARRAY is table of VARCHAR2(10) index by
          BINARY_INTEGER; --Define ENAMES array
    PROCEDURE GetEmpNamesInArray (ArraySize IN INTEGER,
              inEmpnos IN NUMARRAY, outEmpNames OUT VCHAR2ARRAY);
END EmpNames;
/


CREATE PACKAGE BODY EmpNames AS
    PROCEDURE GetEmpNamesInArray (ArraySize IN INTEGER,
              inEmpnos IN NUMARRAY, outEmpNames OUT VCHAR2ARRAY) is
    BEGIN
    FOR I in 1..ArraySize loop
        SELECT ENAME into outEmpNames(I) from EMP
                        WHERE EMPNO = inEmpNos(I);
    END LOOP;
END;

END EmpNames;
/
```

The following example executes the previous procedure to get the `ename` table.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set Empdb = OO4OSession.OpenDatabase("Exampledb", "scott/tiger", 0)
Empdb.Parameters.Add "ArraySize", 3, ORAPARM_INPUT
```

```
Empdb.Parameters.AddTable "EMPNOS", ORAPARM_INPUT, ORATYPE_NUMBER, 3, 22
Empdb.Parameters.AddTable "ENAMES", ORAPARM_OUTPUT, ORATYPE_VARCHAR2, 3, 10
Set EmpnoArray = Empdb.Parameters("EMPNOS")
Set EnameArray = Empdb.Parameters("ENAMES")

'Initialize the newly created input parameter table EMPNOS
EmpnoArray(0) = 7698
EmpnoArray(1) = 7782
EmpnoArray(2) = 7654
Empdb.ExecuteSQL ("Begin EmpNames.GetEmpNamesInArray(:ArraySize," & _
             ":EMPNOS, :ENAMES); End;")
MsgBox EnameArray(0)
MsgBox EnameArray(1)
MsgBox EnameArray(2)
```

See Also:

- Get_Value Method on page 10-167
- Put_Value Method on page 10-220

Executing Data Definition Language Statements

Data Definition Language (DDL) statements manage schema objects in the database. DDL statements create new tables, drop old tables, and establish other schema objects. They also control access to schema objects. For example:

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
EmpDb.ExecuteSQL("create table employees (name VARCHAR2(20)," & _
          "ssn VARCHAR2(12), empno NUMBER(6), mgr NUMBER(6), salary NUMBER(6)")

EmpDb.ExecuteSQL("GRANT UPDATE, INSERT, DELETE ON employees TO donna")
EmpDb.ExecuteSQL("REVOKE UPDATE ON employees FROM jamie")
```

DDL statements also allow you to work with objects in Oracle Database, for example:

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set EmpDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
EmpDb.ExecuteSQL("create type person_t as object (name VARCHAR2(30)," & _
             "ssn VARCHAR2(12),address VARCHAR2(50))")
EmpDb.ExecuteSQL("create table person_tab OF person_t")
```

Transaction Control

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 operation was completed but not the other (for example, if the network connection went down), the bank's 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. The `BeginTrans`, `CommitTrans`, and `Rollback` transactional control

methods of the OraSession object allow operations to be grouped into larger transactions.

The BeginTrans method tells the session that you are starting a group of operations. The CommitTrans method makes the entire group of operations permanent. The Rollback method cancels the entire group. The CommitTrans and Rollback methods end the transaction, and the program returns to normal operation: one transaction for each operation. Experienced Oracle Database users should note the following differences between the operation of Oracle Objects for OLE and many Oracle Database tools:

- Oracle Database 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; they are held until a commit or rollback is executed.
- SQL*Plus starts a new transaction every time a commit or rollback is executed.
- SQL*Plus does not take a row lock in the case of a failed UPDATE or DELETE statement. However, in the case of OO4O, if UPDATE or DELETE methods fail on a given row in a dynaset in a global transaction (such as cases in which you issued a BeginTrans method), be aware that locks remain on those rows. These locks persist until you call a CommitTrans or Rollback method.

If you are connected to more than one database and use the transaction methods, be aware that Oracle Objects for OLE commits each database separately. This is *not* the same as the two-phase commit that Oracle Database provides. If your application needs to guarantee data integrity across databases, connect to a single database and then access additional databases by way of the Oracle Database link feature. This method gives you the benefit of the Oracle Database two-phase commit. Consult your Oracle Database 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 statements). Transactions do not apply to the Data Control Language (DCL) or Data Definition Language (DDL) portions (such as CREATE, DROP, and ALTER statements) of the SQL language. DCL and DDL commands always force a commit, which in turn commits everything done previously.

See Also:

- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- OraConnection Object on page 9-27
- OraSession Object on page 9-58
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235

Microsoft Transaction Server Support

Oracle database transactions initiated in Oracle Objects for OLE (OO4O) automatically participate in global transactions coordinated by the Microsoft Distributed Transaction Coordinator (DTC) in the Microsoft Transaction Server (MTS), if all the following conditions are true:

- The OpenDatabase method of OraSession uses the ORADB_ENLIST_IN_MTS option.

- OO4O determines that it is running in the context of a global transaction in MTS.
- Oracle Service for Microsoft Transaction Server is installed and running.

See Also:

- *Oracle Services for Microsoft Transaction Server Developer's Guide*
- OpenDatabase Method on page 10-212

Asynchronous Processing

In OO4O Automation, you can execute commands using asynchronous processing. This enables you to execute SQL statements and PL/SQL blocks in nonblocking mode. Nonblocking mode is an option of the `CreateSQL` method.

See Also: CreateSQL Method on page 10-111

Nonblocking Mode

In nonblocking mode, control is returned to the application immediately even if the execution is not complete. This allows the application to execute other tasks that are not dependent on the results of the last execution.

To enable nonblocking mode, pass in the `ORASQL_NONBLK` option to the `CreateSQL` method while creating the `OraSQLStmt` object. If this mode is not specified, the `OraSQLStmt` object executes in blocking mode (default behavior).

```
'Create the statement in NON-BLOCKING mode
OraSQL = Oradb.CreateSQL("delete from emp",ORASQL_NONBLK)
```

An `OraSQLStmt` object created in nonblocking mode executes in nonblocking mode for the lifetime of the object.

See Also: OraSQLStmt Object on page 9-60

This section contains the following topics:

Checking the Status of a Nonblocking Operation

To determine the status of an `OraSQLStmt` object executing asynchronously, applications need to poll the `NonBlockingState` property. The `NonBlockingState` property returns `ORASQL_STILL_EXECUTING` if execution is still pending or `ORASQL_SUCCESS` if execution has completed successfully.

Any failures are thrown as exceptions.

On successful completion, the output parameters, if any, are placed in the bound parameter buffers. The application can then access the parameters as in the blocking case.

The following example demonstrates the usage of the `NonBlockingState` property.

```
Dim OraDatabase as OraDatabase
Dim OraStmt as OraSQLStmt
```

```
Dim stat as long
Dim OraSess as OraSession
Set OraSess = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase =OraSess.OpenDatabase("ExampleDb", "scott/tiger", 0)

'execute the select statement with NONBLOCKING mode on
set OraStmt = OraDatabase.CreateSQL ("update emp set sal = sal + 1000", _
            ORASQL_NONBLK)

'Check if the call has completed
stat = OraStmt.NonBlockingState
while stat = ORASQL_STILL_EXECUTING
MsgBox "Asynchronous Operation under progress"
stat = OraStmt.NonBlockingState
wend
MsgBox "Asynchronous Operation completed successfully"
```

See Also: NonBlockingState Property on page 11-111

Canceling a Nonblocking Operation

You can cancel a nonblocking operation that is underway by calling the `Cancel` method on the `OraSQLStmt` object that is executing the asynchronous call.

```
Dim OraDatabase as OraDatabase
Dim OraStmt as OraSQLStmt
Dim stat as long
Dim OraSess as OraSession
Set OraSess = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase =OraSess.OpenDatabase("ExampleDb", "scott/tiger", 0)

'execute the select statement with NONBLOCKING mode on
set OraStmt = OraDatabase.CreateSQL ("update emp set sal = sal + 1000", _
         ORASQL_NONBLK)

'Check if the call has completed
stat = OraStmt.NonBlockingState
if stat = ORASQL_STILL_EXECUTING
MsgBox "Cancelling the asynchronous operation that is underway"
OraStmt.Cancel
End if
```

See Also: Cancel Method on page 10-45

Executing Multiple Queries in Asynchronous Mode

Multiple queries can be executed in asynchronous mode. In this example, while the first connection is executing a non-blocking call, the second connection executes a SQL statement in blocking mode.

```
Dim OraSess as OraSession
Dim OraServ as OraServer
Dim OraDb1 as OraDatabase
Dim OraDb2 as OraDatabase
Dim OraStmtnonblk as OraSQLStmt
Dim OraStmtblk as OraSQLStmt
Dim stat as long
set OraSess = CreateObject("OracleInProcServer.XOraSession")
set OraDb1 = OraSess.OpenDatabase("exampledb","scott/tiger",0&)
```

```
Set OraServ = CreateObject("OracleInProcServer.XOraServer")
set OraDb2 = OraServ.OpenDatabase("Exampledb","scott/tiger",0&)

'execute the select statement with NONBLOCKING mode on
set OraStmtnonblk = OraDb1.CreateSQL ("update emp set sal = sal + 1000", _
             ORASQL_NONBLK)

'Check if the call has completed
stat = OraStmt.NonBlockingState
while stat = ORASQL_STILL_EXECUTING
  MsgBox "Asynchronous Operation under progress"
  stat = OraStmt.NonBlockingState
wend
  MsgBox "Asynchronous Operation completed successfully"

'execute on the second connection in BLOCKING mode
set OraStmtblk = OraDb2.CreateSQL ("update emp set sal = sal + 500",0&)
```

Limitations on Nonblocking

The following are limitations on nonblocking mode:

- When a nonblocking operation is running on an `OraSQLStmt` object, you cannot change the properties or attributes of this object, as it can affect the execution that is in progress.
- You cannot create an `OraSQLStmt` object in nonblocking mode if there are other objects that are already instantiated on the connection. In other words, creating an `OraSQLStmt` object to execute in nonblocking mode only succeeds if no other objects, such as `OraDynaset` and `OraAQ`, are currently active on the same database session. The only exceptions are `OraParameter` and `OraObject` objects. These are permitted, as they may be required for the nonblocking execution.

See Also: "Executing Multiple Queries in Asynchronous Mode" on page 3-17

4

Advanced OO4O Features

This chapter describes advanced Oracle Objects for OLE features.

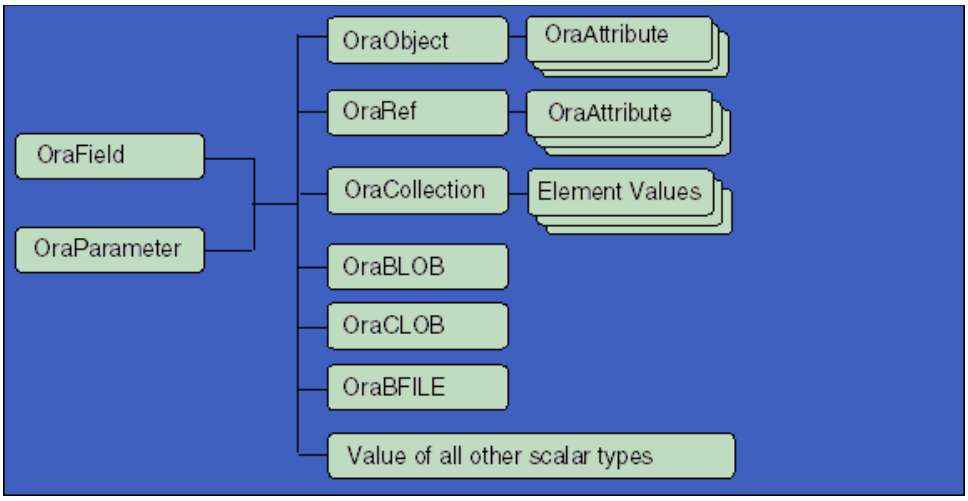
This chapter contains these topics:

Support for Oracle Object-Relational and LOB Data Types

Oracle Objects for OLE provides support for accessing and manipulating instances of REFs, value instances, variable-length arrays (`VARRAY`s), nested tables, and large objects (LOBs) in an Oracle database.

Table 4–1 illustrates the containment hierarchy for instances of all types in Oracle Objects for OLE.

Figure 4–1 Object-Relational and LOB Data Types Diagram



****

Instances of these types can be fetched from the database or passed as input or output variables to SQL statements and PL/SQL blocks, including stored procedures and functions. All instances are mapped to COM Automation interfaces that provide methods for dynamic attribute access and manipulation. These interfaces can be obtained from:

- The `Value` property of an `OraField` object in a dynaset.
- The `Value` property of an `OraParameter` object used as an input or an output parameter in SQL Statements or PL/SQL blocks.
- An attribute of another object/`REF` instance.
- An element in a collection (`VARRAY` or a nested table).

Instantiating Oracle LOBs, Objects, and Collections

Oracle Objects for OLE provides COM Automation interfaces for working with LOBs, Oracle objects, and collection types. These interfaces provide methods and properties to access data associated with LOBs, Oracle objects, and collection instances.

Oracle LOBs, Objects, and Collections

Table 4–1 lists Oracle LOBs, Objects, and collection types with associated OO4O interfaces.

Table 4–1 Oracle LOBs, Objects, and Collections

Type	OO4O Interface
`Object`	`OraObject`
`REF`	`OraRef`
`VARRAY` and Nested Table	`OraCollection`
`BLOB`	`OraBlob`

Table 4–1 (Cont.) Oracle LOBs, Objects, and Collections

Type	OO4O Interface
CLOB	OraClob
BFILE	OraBFile

How the preceding interfaces are retrieved in OO4O depend on how they are stored in the database or accessed in a SQL statement. These are the possible scenarios:

- Column of a table

  If a table contains LOBs, object types, and collections as columns and the dynaset SELECT statement is based on this table, then the Value property of the OraField object representing that column returns corresponding OO4O interfaces for that type.

- Bind variable in a SQL statement or PL/SQL block

  If a SQL statement or PL/SQL block has LOBs, object types, and collections as bind variables, then an OraParameter object should be created with a corresponding server type using the Add method. The Value property of the OraParameter object representing that bind variable returns the corresponding OO4O interfaces for that type.

- Attribute of an Oracle object instance

  If an Oracle object instance has LOBs, object types, or collections as attributes, then the corresponding OO4O interface for any attribute is retrieved by using the subscript or name of the attribute from the OraObject or OraRef, or by using the Value property of an OraAttribute object.

- Element of VARRAY and nested table

  If an Oracle VARRAY and nested table has object types and REF as its elements, then the corresponding OO4O interface is retrieved using the element index as the subscript from the OraCollection object.

When OO4O interfaces for these types are retrieved as part of a dynaset, then the OO4O interfaces represent instances of LOBs, objects, and collection types for the *current row* of the dynaset. If the current row changes due to a move operation, then the OO4O interfaces represent instances of LOBs, objects, and collection types for the new current row. When OO4O interfaces for these types are retrieved as part of an OraParameter object and the OraParameter value changes to due to a OraSQLStmt Refresh method, then the OO4O interface represents a new instance LOB, object, and collection type for that OraParameter.

Internally, OO4O maintains one OO4O interface for each OraField, OraParameter, and OraAttribute object. To retain the instance of LOBs, objects, and collection types independent of a dynaset move operation or an OraSQLStmt refresh operation, use the Clone method on the corresponding OO4O interface. This method makes a copy of LOBs, objects, and collection types instance and returns a corresponding OO4O interface associated with that copy.

Using Large Objects (LOBs)

The large object (LOB) data types (BLOB, CLOB, NCLOB, and BFILE) can provide storage for large blocks of unstructured data, such as text, images, video clips, and sound waveforms, up to 4 gigabytes in size. They provide efficient, random,

piece-wise access to the data. In Oracle Objects for OLE, instances of LOB data types are represented as interfaces.

See Also:

- OraBLOB, OraCLOB Objects on page 9-11
- OraBFILE Object on page 9-9
- "Schema Objects Used in LOB Data Type Examples" on page A-3 for schema objects used in the OraLOB and BFILE examples

This section includes the following topics:

- LOB Data Types
- Using OraBLOB and OraCLOB
- Retrieving LOBs From the Database
- Performance Considerations with LOB Read and Write
- Writing LOB Data
- Reading LOB Data

LOB Data Types

Table 4–2 lists the four LOB data types and their corresponding OO4O interfaces.

Table 4–2 LOB Data Types

LOB Data Types	a LOB whose value is composed of	Corresponding OO4O Interface
BLOB	Unstructured binary (raw) data.	OraBLOB
CLOB	Fixed-width, single-byte character data that corresponds to the database character set defined for Oracle Database.	OraCLOB
NCLOB	Fixed-width, multiple-byte character data that corresponds to the national character set defined for Oracle Database.	OraCLOB
BFILE	A LOB whose large binary data is stored in operating system files outside of database tablespaces. BFILEs can also be located on tertiary storage devices such as hard disks, CD-ROMs, Photo CDs, and DVDs.	OraBFILE

The following example creates a table that has BLOB and CLOB columns, and inserts rows into the table using the ExecuteSQL method on an OraDatabase object.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")

Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
InvDb.ExecuteSQL("create table part(part_id NUMBER, part_name
VARCHAR2(20),part_image BLOB, part_desc CLOB)")
InvDb.ExecuteSQL ("insert into part values (1,'ORACLE NETWORK',EMPTY_BLOB()," & _
              "EMPTY_CLOB())")
InvDb.ExecuteSQL ("insert into part values (2,'ORACLE SERVER', EMPTY_BLOB()," & _
           "EMPTY_CLOB())")
```

The EMPTY_BLOB() and EMPTY_CLOB() PL/SQL functions provide an empty LOB to insert into the LOB column.

See Also: ExecuteSQL Method on page 10-144

Using OraBLOB and OraCLOB

OraBLOB and OraCLOB interfaces in OO4O provide methods for performing operations on large objects in the database including BLOB, CLOB, and NCLOB, and BFILE data types.

The following Visual Basic example illustrates how to read the PartImage from the part table:

```
Dim Buffer as Variant
Set Part = OraDatabase.CreateDynaset("select * from part", 0&)
set PartImage = OraDynaset.Fields("part_image").Value

'read the data into the buffer
amount_read = PartImage.Read(buffer)

'copy the image content into the file
PartImage.CopyToFile "d:\image\partimage.jpg"
```

See Also: OraBLOB, OraCLOB Objects on page 9-11

Retrieving LOBs From the Database

OraBlob, OraClob, and OraBFile objects can be retrieved using an OraDynaset object or a parameter object:

Using an OraDynaset Object

If a table contains a LOB column and a dynaset query selects against that LOB column, then the Value property of the OraField object returns a OraBlob, OraClob, or a OraBFile object.

The following example selects LOB columns from the part table. PartDesc and PartImage are OraBlob and OraClob objects that are retrieved from the OraField object.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
Set Part = InvDb.CreateDynaset("select * from part", 0&)
Set PartDesc = Part.Fields("part_desc").Value
Set PartImage = Part.Fields("part_image").Value
```

Using a Parameter object

If a SQL statement or PL/SQL block has a bind variable of type LOB, you create a OraParameter object using the OraParameters Add method. The Value property of the OraParameter object for that bind variable returns an OraBlob, OraClob, or OraBFile object.

The following example illustrates how to use a LOB data type as a bind variable in a PL/SQL anonymous block. This block selects a LOB column from the database.

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
InvDb.Parameters.Add "PartDesc", Null, ORAPARM_OUTPUT,ORATYPE_CLOB
```

```
InvDb.Parameters.Add "PartImage", Null, ORAPARM_OUTPUT,ORATYPE_BLOB
InvDb.ExecuteSQL ("BEGIN select part_desc, part_image into :PARTDESC," & _
            ":PARTIMAGE from part where part_id = 1 for update NOWAIT; END;") & _
            "for update NOWAIT; END;")
Set PartDesc = InvDb.Parameters("PartDesc").Value
Set PartImage = InvDb.Parameters("PartImage").Value
```

Performance Considerations with LOB Read and Write

When reading and writing LOBs, there are several options that can optimize an application's memory usage and reduce the number of network round-trips.

Single-Piece Operation

The contents of a buffer are read or written to the database in one round-trip.

Multiple-Piece Operation

A small buffer is used for multiple calls to read or write methods. In this mode, the data is streamed, rather than requiring a complete round-trip for each read or write call. This method is quicker than doing several small single-piece operations. It has the restriction that the data must be read and written sequentially, meaning that the offset increases automatically with each read or write. The total amount must be known before it is written, and the operation cannot be aborted before completion.

See Also:

- OraBLOB, OraCLOB Objects on page 9-11
- Read (OraLOB/BFILE) Method on page 10-221
- Write (OraLOB) Method on page 10-261

LOB Buffering Option

The LOB buffering option automatically buffers any read or write operations. A network round-trip occurs only when the `FlushBuffer` method is called. This is most useful when there are many small writes that occur all across the LOB. This method has significant restrictions.

See Also: EnableBuffering (OraLOB) Method on page 10-139

Writing LOB Data

The `Write` method of the `OraBlob` and `OraClob` objects writes data from a local buffer to a LOB in the database. The `CopyFromFile` (OraLOB) method writes content of a local file to a LOB in the database.

Any operation that changes the value of a LOB, including the `Write` method, can only occur when the row the LOB is associated with has been locked. If a LOB field is null, it must first be updated with an empty LOB before a method can write to the LOB field.

LOB data can be written in one piece or in a series of multiple pieces., as described in the following topics:

- Single-Piece Write Operation
- Multiple-Piece Write Operation

See Also:

- Write (OraLOB) Method on page 10-261
- CopyFromFile (OraLOB) Method on page 10-73

Single-Piece Write Operation

The entire contents of a buffer can be written in a single piece in one network round-trip. The following example writes 10 KB of data from the local file `partimage.dat` to `part_image` column at the offset of `1000`.

```
Dim buffer() as byte
ReDim buffer(10000)
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
Set Part = InvDb.CreateDynaset("select * from part", 0&)
Set PartImage = Part.Fields("part_image").Value
PartImage.Offset = 1000
FNum = FreeFile
Open "PartImage.Dat" For Binary As #FNum
Get #FNum, , buffer
Part.Edit

amount_written = PartImage.Write(buffer)
Part.Update
Close FNum
```

The `CopyFromFile` (OraLOB) method writes data directly to a LOB from a local file. The following code is functionally the same as the previous code:

```
Part.Edit
PartImage.CopyFromFile "PartImage.dat" , 10000, 1000
Part.Update
```

See Also: CopyFromFile (OraLOB) Method on page 10-73

Multiple-Piece Write Operation

This mechanism is used when the size of the buffer available is smaller than the total amount of data to be written. The total amount of data to be written is set by using the `PollingAmount` (OraLOB/`BFILE`) property.

The `Offset` (OraLOB/`BFILE`) property is used only once to set the offset for the first piece `Write` operation. After the first time, it is automatically increased by the size of the previous piece. The `Status` (OraLOB/`BFILE`) property must be checked for success of each piece `Write` operation. If the `Status` property returns `ORALOB_NEED_DATA`, the `Write` method must be called again. This must continue until the amount specified by the `PollingAmount` property has been sent.

The `piecetype` argument of the `Write` method must be set to `ORALOB_FIRST_PIECE` for the first piece that is sent, and last piece `Write` operation ends with setting the `piecetype` argument to `ORALOB_LAST_PIECE`. At the end of multiple piece operation, the `Status` property returns `ORALOB_NO_DATA`.

The following example writes 102 KB of data in 10 KB chunks to the `part_image` column from the local file `partimage.dat` at offset of `1000`.

```
Dim buffer() as byte
chunksize = 10000
ReDim buffer(chunksize)
```

```
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
Set Part = InvDb.CreateDynaset("select * from part", 0&)
Set PartImage = Part.Fields("part_image").Value

FNum = FreeFile
Open "PartImage.Dat" For Binary As #FNum
PartImage.Offset = 1000
PartImage.PollingAmount =102000
remainder = 102000
Part.Edit
Get #FNum, , buffer
amount_written = PartImage.Write(buffer, chunksize, ORALOB_FIRST_PIECE)

While PartImage.Status = ORALOB_NEED_DATA
remainder = remainder - chunksize
If remainder < chunksize Then
piecetype = ORALOB_LAST_PIECE
chunksize = remainder

Else
piecetype = ORALOB_NEXT_PIECE
End If
Get #FNum, , buffer
amount_written = PartImage.Write(buffer, chunksize, piecetype)
Wend
Close FNum
Part.Update
```

See Also:

- PollingAmount Property on page 11-125
- Offset (OraLOB/BFILE) Property on page 11-112
- Status (OraLOB/BFILE) Property on page 11-154

Reading LOB Data

The `OraBlob` and `OraClob` `Read` method reads data to a local buffer from a LOB in the database. The `CopyFromFile` method reads the contents of a LOB into a local file.

LOB data can be read in one piece or in a series of multiple pieces, as described in the following topics:

- Single-Piece Read Operation
- Multiple-Piece Read Operation

See Also: Read (OraLOB/BFILE) Method on page 10-221

Single-Piece Read Operation

The entire contents of a buffer can be read in a single piece in one network round-trip. The following example reads 10 KB of data from the `part_image` column at an offset of `1000` to the local file `image.dat`.

```
Dim buffer as Variant
Dim buf() As Byte
chunksize = 10000
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
```

```
Set Part = InvDb.CreateDynaset("select * from part", 0&)
Set PartImage = Part.Fields("part_image").Value
FNum = FreeFile
Open "image.dat" For Binary As #FNum
PartImage.Offset = 1000
amount_read = PartImage.Read(buffer,10000)
buf = buffer
Put #FNum, , buf
Close FNum
```

The `CopyToFile` (OraLOB/`BFILE`) method writes data directly to a local file from a LOB. The following code is functionally the same as the previous code:

```
PartImage.CopyToFile "image.dat" , 10000, 1000
```

See Also: CopyToFile (OraLOB/BFILE) Method on page 10-76

Multiple-Piece Read Operation

This mechanism is used when the size of the buffer available is smaller than the total amount of data to be read. The total amount of data to be read is set by using the `PollingAmount` (OraLOB/`BFILE`) property. The `Offset` (OraLOB/`BFILE`) property is used only once to set the offset for the first piece `Read` operation. After the first time, it is automatically increased by the size of the previous piece.

The `Status` (OraLOB/`BFILE`) property must be checked for success for each piece `Read` operation. If the `Status` property returns `ORALOB_NEED_DATA`, the `Read` method must be called again. This must continue until the amount specified by the `PollingAmount` property has been read. At the end of multiple piece operations, the `Status` property returns `ORALOB_NO_DATA`.

The following example reads 102 KB of data in 10 KB chunks from the `part_image` column at offset of `1000` to the local file `image.dat`.

```
Dim buffer as Variant
Dim buf() As Byte
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
Set InvDb = OO4OSession.OpenDatabase("INVDB", "scott/tiger", 0)
Set Part = InvDb.CreateDynaset("select * from part", 0&)
Set PartImage = Part.Fields("part_image").Value
FNum = FreeFile
Open "image.dat" For Binary As #FNum
PartImage.offset = 1000
PartImage.PollingAmount = 102000
amount_read = PartImage.Read(buffer, chunksize)
buf = buffer
Put #FNum, , buf
While PartImage.Status = ORALOB_NEED_DATA
amount_read = PartImage.Read(buffer, chunksize)
buf = buffer
Put #FNum, , buf
Wend
Close FNum
```

See Also:

- PollingAmount Property on page 11-125
- Offset (OraLOB/BFILE) Property on page 11-112
- Status (OraLOB/BFILE) Property on page 11-154

Oracle Object Data Types

An object type is a user-defined composite data type created in the database. A column can represent an object type or a row can represent an object type. An instance of the `Object` type can be stored in the database. This object instance can be fetched to the client side and modified using Oracle Objects for OLE.

See Also: *Oracle Database Object-Relational Developer's Guide*

There are two types of object instances.

- `OraObject` object

  If a column represents an object type, then an instance of this object type is referred to as an embedded instance or a value instance. In OO4O, this type is represented by an `OraObject` object. For example, an `ADDRESS` object type is stored as a column in the `PERSON` table. `OraObject` objects can be embedded within other structures. An embedded instance or a value instance can also be the attributes of another object instance.

  See Also: "About the OraObject Interface" on page 4-11

- `OraRef` object

  If a row in an object table represents an object type, then the instance of this type is referred to as a referenceable object. In OO4O, this type is represented by an `OraRef` object. An internally referenceable object has a unique object identifier that is represented by the `REF` data type. A `REF` column can be thought of as a pointer to a referenceable object. OO4O applications can retrieve a `REF` data type from a referenceable object, fetch (pin) the associated referenceable object to the client side, and update (flush) the modified referenceable object to the database.

  See Also: "About the OraRef Interface" on page 4-13

About the OraObject Interface

The `OraObject` interface is a representation of an Oracle embedded object or a value instance. It contains a collection interface (`OraAttributes`) for accessing and manipulating (updating and inserting) individual attributes of a value instance.

Individual attributes of an `OraAttributes` collection interface can be accessed by using a subscript or the name of the attribute.

The following Visual Basic example illustrates how to access attributes of the `Address` object in the `person_tab` table:

```
Set Person = OraDatabase.CreateDynaset("select * from person_tab",0&)
set Address = Person.Fields("Addr").Value
msgbox Address.Zip
msgbox.Address.City
```

See Also: OraObject Object on page 9-43

Using the OraObject Interface

The following example creates an ADDRESS object type having street, city, state and zip as its attributes and a PERSON table having an ADDRESS object type column. It also inserts data using the ExecuteSQL method of the OraDatabase object.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
HRDb.ExecuteSQL("create type ADDRESS as object ( street
VARCHAR2(200), city varchar2(20), state CHAR(2), zip varchar2(10) )")
HRDb.ExecuteSQL("create table person (name varchar2(20), age number," & _
                "addr ADDRESS) ")
HRDb.ExecuteSQL("insert into person values('nasser',40, " & _
                "address('Wine Blvd', 'Pleasanton', 'CA', '94065'))")
HRDb.ExecuteSQL("insert into person values('Maha', 25," & _
                "address('Continental Way', 'Belmont', 'CA', '94002'))")
HRDb.ExecuteSQL("insert into person values('chris',30, address('First " & _
                "Street', 'San Francisco', 'CA' ,'94123'))")
```

The following topics discuss manipulating the OraObject interface:

- Retrieving an Embedded/Value Instance from the Database
- Accessing Attributes of an Embedded/Value Instance
- Modifying Attributes of an Embedded/Value Instance

See Also: ExecuteSQL Method on page 10-144

Retrieving an Embedded/Value Instance from the Database

An OraObject object can be retrieved using OO4O using a dynaset or parameter object:

Using a Dynaset Object If a table contains an object type column and a dynaset query selects against that column, then the Value property of the OraField object returns an OraObject.

The following code selects an ADDRESS column from the person table, and then an Address object is retrieved from the OraField object.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Person = hrDb.CreateDynaset("select * from person", 0&)
set Address = Person.Fields("Addr").Value
```

Using a Parameter Object If a SQL statement or a PL/SQL block has a bind variable of object type, you create an OraParameter object using the OraParameters Add method. The Value property of the OraParameter object for that bind variable returns an OraObject object.

The following example uses an object data type as a bind variable in a PL/SQL anonymous block. This block selects an object column from the database.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
hrDb.Parameters.Add "ADDRESS", Null, ORAPARM_OUTPUT, ORATYPE_OBJECT, "ADDRESS"
'execute the sql statement which selects Address from the person_tab
hrDb.ExecuteSQL ("BEGIN select Addr into :ADDRESS from person where " & _
```

```
            "age = 40; end;")
'retrieve Address object from the OraParameter
set address = hrDb.Parameters("ADDRESS").Value
```

See Also:

- OraObject Object on page 9-43
- OraField Object on page 9-33
- OraParameter Object on page 9-50

Accessing Attributes of an Embedded/Value Instance

Individual attributes can be accessed by using a subscript or the name of the attribute. The following example illustrates how to access attribute values of an `ADDRESS` object instance.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Person = hrDb.CreateDynaset("select * from person", 0&)
set Address = Person.Fields("Addr").Value
msgbox Address.City
msgbox Address.Street
msgbox Address.State
msgbox Address.Zip
```

The following code accesses all of the attribute values:

```
For I=1 to Address.Count
   msgbox Address(I)
Next I
```

Modifying Attributes of an Embedded/Value Instance

If the object instance is retrieved using a dynaset object, its attribute values can be modified between a dynaset `Edit/Update` pair. The following example modifies the street and city attribute values of the `ADDRESS` object instance.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Person = hrDb.CreateDynaset("select * from person", 0&)
set Address = Person.Fields("Addr").Value
Person.Edit
      Address.Street  =  "Oracle Parkway"
      Address.City = "Redwood shores"
Person.Update
```

Executing a Member Method of an Oracle Object Instance

Oracle object type member methods are created during type creation. Oracle object instance member methods are executed in OO4O as PL/SQL procedures or functions. Arguments and return values to the member methods should be bound using the `OraParameter` object. The first argument to the member method should always be the object instance. This object instance can be bound with the `ORAPARM_INPUT` or `ORAPARM_BOTH` mode. If the member method modifies the attributes of an object instance and a new object instance needs to be retrieved to the OO4O application, then this object instance must be bound with the `ORAPARM_BOTH` mode.

For example, if a bank_account object type has open, close, and deposit as member methods, then the schema for the bank_account object type is the following:

```
CREATE OR REPLACE TYPE bank_account AS OBJECT (
    acct_number INTEGER(5),
    balance REAL,
    MEMBER PROCEDURE open (amount IN REAL),
    MEMBER PROCEDURE close (num IN INTEGER, amount OUT REAL),
    MEMBER PROCEDURE deposit (SELF IN OUT bank_bccount,num IN
                        INTEGER, amount IN REAL),
);
```

In OO4O, BankObj is an OraObject object representing a valid bank object instance from the database. To execute the deposit method, the SELF, num, and amount arguments need to be bound using the OraParameter object.

```
Dim BankObj as OraObject
assumes that we have valid BankObj
set BankObj = .....

'create a OraParameter object for bank_account object and set it to BankObj
OraDatabase.Parameters.Add "BANK", BankObj, ORAPARM_BOTH, ORATYPE_OBJECT, _
                     "BANK_ACCOUNT"

'create a OraParameter object for num argument and set the value to 100
OraDatabase.Parameters.Add "ACCOUNT_NO", 100, ORAPARM_INPUT, ORATYPE_NUMBER

'create a OraParameter object for amount argument and set the value to 1200
OraDatabase.Parameters.Add "AMOUNT", 1200, ORAPARM_OUTPUT, ORATYPE_NUMBER

'display the balance from the bank object
Bankobj.balance

'now execute the PL/SQL block for member method execution
OraDatabase.ExecuteSQL ("BEGIN BANK_ACCOUNT.DEPOSIT :BANK," & _
                    (":ACCOUNT_NO,:AMOUNT); END;")

'get the modified bank object from the parameter
set Bankobj  = OraDatabase.Parameters("BANK").Value

'display the new balance
Bankobj.balance
```

About the OraRef Interface

The OraRef interface represents an instance of a referenceable object (REF) in client applications. The object attributes are accessed in the same manner as attributes of an object represented by the OraObject interface. The OraRef interface is derived from an OraObject interface through the containment mechanism in COM. REF objects are updated and deleted independently of the context from which they originated, such as dynasets. The OraRef interface also encapsulates the functionality for navigating through graphs of objects utilizing the Complex Object Retrieval Capability (COR) in Oracle Call Interface (OCI).

See Also: "OraRef Object" on page 9-52

Using the OraRef Interface

This section demonstrates the creation of an object table named PERSON_TAB. The object table is based on the object type PERSONOBJ. Each reference to the rows of this object table is stored in an aperson REF type column of the CUSTOMERS table. The following code creates database schemas:

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
HRDb.ExecuteSQL("create type PERSONOBJ  as object ( name varchar2(20), " & _
                "age number, addr ADDRESS)")
HRDb.ExecuteSQL("create table person_tab of personobj")
HRDb.ExecuteSQL("insert into person_tab values('nasser',40," & _
                 "address('Wine Blvd', 'Pleasanton', 'CA', '94065'))")
HRDb.ExecuteSQL("insert into person_tab values('Maha', 25, " & _
                 "address('Continental Way', 'Belmont', 'CA', '94002'))")
HRDb.ExecuteSQL("insert into person_tab values('chris',30, " & _
               "address('First Street', 'San Francisco', 'CA' , '94123'))")
```

The following code creates a CUSTOMERS table having an aperson REF column referencing rows of the object table:

```
HRDb.ExecuteSQL("create table CUSTOMERS (account number,
aperson REF personobj)")

HRDb.ExecuteSQL("insert into customers values(10, null)")
HRDb.ExecuteSQL("insert into customers values(20, null)")
HRDb.ExecuteSQL("insert into customers values(30, null)")
HRDb.ExecuteSQL("update customers set aperson = (select ref(p) from " & _
                "person_tab p where p.name = 'nasser') where account = 10")
HRDb.ExecuteSQL("update customers set aperson = (select ref(p) from " & _
                "person_tab p where p.name = 'Maha') where account = 20")
HRDb.ExecuteSQL("update customers set aperson = (select ref(p) from " & _
                "person_tab p where p.name = 'chris') where account = 30")
```

The following topics discuss manipulating the OraRef Interface:

- Retrieving a REF from the Database
- Accessing Attributes of a Referenceable Instance
- Modifying Attributes of a Referenceable Instance

See Also: OraRef Object on page 9-52

Retrieving a REF from the Database

An OraRef object can be retrieved using OO4O in the following ways:

Using a Dynaset Object If a table contains a REF type column and a dynaset query selects against that column, then the Value property of the OraField object returns an OraREF.

The following example selects an aperson column from the person table, and the aperson object is retrieved from the OraField object.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Customer = hrDb.CreateDynaset("select * from customers", 0&)
set Person = Customer.Fields("aperson").Value
```

Using an OraParameter Object If a SQL statement or PL/SQL block has a bind variable of REF type, you create an OraParameter object using the OraParameters Add method. The Value property of the OraParameter object for that bind variable returns an OraREF.

The example illustrates using a REF object data type as a bind variable in a PL/SQL anonymous block. The block selects an object column from the database.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
hrDb.Parameters.Add "PERSON", Null, ORAPARM_OUTPUT, ORATYPE_REF,"PERSONOBJ"

'execute the sql statement which selects Address from the person_tab
hrDb.ExecuteSQL ("BEGIN select aperson into :PERSON from customers" & _
                 "where account = 10; end;")

'retrieve Person object from the OraParameter
set Person = hrDb.Parameters("PERSON").Value
```

See Also: OraRef Object on page 9-52

Accessing Attributes of a Referenceable Instance

Before accessing attributes of a referenceable instance, it should be fetched (pinned) on the client side. OO4O implicitly pins the REF value when attribute values are accessed from the OraRef object. After the pin operation, attributes of the referenceable instance are accessed in the same manner as attributes of a value instance represented by the OraObject object.

The following example pins the APERSON REF value (implicitly) and accesses its name and address attributes. Note that accessing the address attribute returns an Address OraObject object.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Customer = hrDb.CreateDynaset("select * from customers", 0&)
set Person = Customer.Fields("APERSON").Value
msgbox Person.Name
set Address = Person.Addr
msgbox Address.City
```

See Also: OraRef Object on page 9-52

Modifying Attributes of a Referenceable Instance

Because a referenceable instance is stored in a row of an object table, modifying attributes of referenceable instance requires an object lock. Therefore, rows corresponding to the object instance in an object table should be locked, which can be done by calling the Edit method of the OraRef object. The OraRef Update method releases the object lock.

The following example modifies the age attribute of Person object.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
set Customer = hrDb.CreateDynaset("select * from customers", 0&)
set Person = Customer.Fields("APERSON").Value
Person.Edit
Person.Age = 45
Person.Update
```

See Also:

- OraRef Object on page 9-52
- Update (OraRef) Method on page 10-259
- Edit (OraRef) Method on page 10-136

Oracle Collections

A collection is an ordered group of elements, all of the same type. Each element has a unique subscript, called an index, that determines its position in the collection.

Note: An `OraCollection` element index starts at `1`.

A collection can be subdivided into the following types:

- Nested table type

  Viewed as a table stored in the column of a database table. When retrieved, the rows of a nested table are given consecutive subscripts starting at 1, and individual rows are accessed using array-like access.

- `VARRAY` type

  Viewed as an array stored in the column of a database table. To reference an element in a `VARRAY` type, standard subscripting syntax can be used. For example, `Grade(3)` references the third element in `VARRAY Grades`.

In Oracle Objects for OLE, an Oracle collection type is represented by the `OraCollection` interface. The following topics provide more information:

- About the OraCollection Interface
- Retrieving a Collection Type Instance from the Database
- Accessing Collection Elements
- Modifying Collection Elements
- Creating a VARRAY Collection Type
- Creating a Dynaset from an OraCollection Object

See Also:

- "OraCollection Object" on page 9-19
- "Schema Objects Used in OraCollection Examples" on page A-3

About the OraCollection Interface

The `OraCollection` interface provides methods for accessing and manipulating Oracle collection types, namely variable-length arrays (`VARRAY`s) and nested tables in OO4O. Elements contained in a collection are accessed by subscripts.

The following Visual Basic example illustrates how to access attributes of the `EnameList` object from the department table:

```
Set Person = OraDatabase.CreateDynaset("select * from department",0&)
set EnameList = Department.Fields("Enames").Value

'access all elements of the EnameList VArray
```

```
for I=1 to I=EnameList.Size
   msgbox EnameList(I)
Next I
```

See Also: OraCollection Object on page 9-19

Retrieving a Collection Type Instance from the Database

A collection type can be retrieved using OO4O in the following ways:

Using a Dynaset Object

If a table contains a collection type column and a dynaset query selects against that column, then the `Value` property of the `OraField` object returns an `OraCollection` object.

The following example selects the `ENAMES` column from the `department` table, and an `EnameList` object is retrieved from the `OraField` object:

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
Set Dept = hrDb.CreateDynaset("select * from department", 0&)
Set EnameList = Dept.Fields("ENAMES").Value
```

Using a Parameter Object

If a SQL statement or PL/SQL block has a bind variable of collection type, then you create a `OraParameter` object using the `OraParameters Add` method. The `Value` property of the `OraParameter` object for that bind variable returns an `OraCollection` object.

The following example uses a collection data type as a bind variable in a PL/SQL anonymous block and selects a collection type from the database:

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
hrDb.Parameters.Add "ENAMES", Null, ORAPARM_OUTPUT, ORATYPE_VARRAY, "ENAMELIST"
hrDb.ExecuteSQL ("BEGIN select enames into :ENAMES from department" & _
              "where dept_id = 10; END;")
set EnameList = hrDb.Parameters("ENAMES").Value
```

See Also:

- OraCollection Object on page 9-19
- OraField Object on page 9-33

Accessing Collection Elements

Individual element values are accessed by using a subscript. For example, the `Value` returned by the `OraCollection` object for subscript `1` is the element value at index `1`. The maximum value of the subscript is equal to the total number of elements in the collection including any deleted elements. The `OraCollection` subscript starts from 1.

The following example code retrieves the `Enamelist` collection instance and accesses its elements at the first and second index.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
```

```
Set Dept = hrDb.CreateDynaset("select * from department", 0&)
Set EnameList = Dept.Fields("ENAMES").Value
msgbox EnameList(1)
msgbox EnameList(2)
```

This code displays all the element values of the `EnameList` collection.

```
For I = 1 to EnameList.Size
   msgbox EnameList(I)
Next I
```

See Also: OraCollection Object on page 9-19

Modifying Collection Elements

If the collection instance is retrieved using a dynaset object, element values can be modified between a dynaset `Edit` and `Update` pair. The following example code modifies the second element value of an `Enamelist` collection instance.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)

Set OraDynaset = hrDb.CreateDynaset("select * from department", 0&)
Set EnameList = OraDynaset.Fields("ENAMES").Value

OraDynaset.Edit
    EnameList(2) = "Chris"
OraDynaset.Update
```

Creating a VARRAY Collection Type

The example code that follows creates a `VARRAY` collection type `ENAMELIST` and a `department` table having `ENAMELIST` collection type column.

```
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set hrDb = OO4OSession.OpenDatabase("Exampledb", "scott/tiger", 0)
hrDb.ExecuteSQL("CREATE TYPE ENAMELIST AS VARRAY(20) OF VARCHAR2(30)")
hrDb.ExecuteSQL("CREATE TABLE department (dept_id NUMBER(2),name" & _
               "VARCHAR2(15),ENAMES ENAMELIST)")
```

The following script inserts some collection data into `department` table:

```
hrDb.ExecuteSQL("INSERT INTO department VALUES(10, 'ACCOUNTING'," & _
               "ENAMELIST('KING','CLARK','MILLER') )")
hrDb.ExecuteSQL("INSERT INTO department VALUES(20, 'RESEARCH'," & _
               "ENAMELIST('JONES','SCOTT','ADAMS','SMITH','FORD') )")
hrDb.ExecuteSQL("INSERT INTO department VALUES(30, 'SALES'," & _
               "ENAMELIST('BLAKE','MARTIN','ALLEN','TURNER','JAMES') )")
```

See Also: OraCollection Object on page 9-19

Creating a Dynaset from an OraCollection Object

A `SELECT` query can be issued against instances of the `VARRAY` and nested table collection types using SQL `THE` or `TABLE` operators and individual elements can be accessed as rows. If these collection types have object types for element types, then individual attributes of the object type represents fields of a row.

For example, if an object type X has attributes a, b, and c, and the element type of the collection is object type X, then the SELECT query on this collection returns a, b, and c fields.

In OO4O, read-only dynaset objects can be created from SELECT queries on the collection. Individual elements are accessed using row navigation. If the collection type has an object type as its element type, then attributes of that object type (element) are accessed using the OraField object.

This discussion assumes you have a Course object type and a CourseList nested table collection type with Course as its element type, as described here:

```
CREATE TYPE Course AS OBJECT (
    course_no NUMBER(4),
    title VARCHAR2(35),
    credits NUMBER(1)
);
CREATE TYPE CourseList AS TABLE OF Course;
```

In OO4O, CourseList OraCollection represents an instance of the CourseList collection type.

```
Dim CourseList as OraCollection
```

Assume that you have valid a CourseList collection instance:

```
set CourseList = ......
```

The SQL THE or TABLE operator needs collection type as a bind variable. Create a OraParameter object for the CourseList OraCollection as follows:

```
OraDatabase.Parameters.Add "COURSELIST", CourseList, ORAPARM_INPUT, _
            ORATYPE_TABLE, "COURSELIST"
```

Create a read-only dynaset based on the CourseList using the SQL THE operator:

```
Set CourseListDyn = OraDatabase.CreateDynaset("select * from THE (select" & _
           "CAST(:COURSELIST AS COURSELIST) from dual)", ORADYN_READONLY)
```

You can also create a read-only dynaset based on the CourseList using the SQL TABLE operator, which is available only in OO4O with libraries from release Oracle9*i* and on:

```
Set CourseListDyn = OraDatabase.CreateDynaset("select * from" & _
          "TABLE(CAST(:COURSELIST AS COURSELIST))", ORADYN_READONLY)

'display the course_no field
msgbox CourseListDyn.Fields("course_no").Value

'display the title field
msgbox CourseListDyn.Fields("title").Value

'move to next row
OraDynaset.MoveNext
```

See Also: OraCollection Object on page 9-19

Example: Creating a Dynaset from an OraCollection Object

The following example illustrates how to create a dynaset from an OraCollection object. Before running the sample code, make sure that you have the necessary data

types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim CourseList as OraCollection
Dim Course as OraObject
Dim CourseListDyn as OraDynaset

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

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

'create a dynaset object from division
set OraDynaset = OraDatabase.CreateDynaset("select * from division", 0&)

'retrieve a Courses column from Division. Here Value property
'of OraField object 'returns CourseList OraCollection
set CourseList = OraDynaset.Fields("Courses").Value

'create a input parameter for CourseList for nested table dynaset
OraDatabase.Parameters.Add "COURSELIST", CourseList, ORAPARM_INPUT, _
                ORATYPE_TABLE, "COURSELIST"

'create a read only dynaset based on the CourseList.
Set CourseListDyn = OraDatabase.CreateDynaset("select * from" & _
                "THE(select CAST(:COURSELIST AS COURSELIST) from dual)", _
                ORADYN_READONLY)

'dynaset can also be created from Oracle8 collection using the
'following statement
'Set CourseListDyn = OraDatabase.CreateDynaset("select * from
'TABLE(CAST(:COURSELIST AS COURSELIST))", ORADYN_READONLY)

'get the field values of the collection dynaset
msgbox CourseListDyn.Fields("title").Value
msgbox CourseListDyn.Fields("course_no").Value

'move the original dynaset to second row
Oradynaset.MoveNext

'set the new value of CourseList  collection from the second row
'of main dynaset to the "COURSELIST" parameter
OraDatabase.Parameters("COURSELIST").Value = CourseList

'refresh the collection dynaset. Now the collection dynaset values are refreshed
'with new collection value. CourseListDyn.Refresh
'get the field values of the collection dynaset
msgbox CourseListDyn.Fields("title").Value
msgbox CourseListDyn.Fields("course_no").Value
```

Advanced Queueing Interfaces

Oracle Objects for OLE provides the `OraAQ` Automation interface with methods for enqueuing and dequeuing messages. The `OraAQMsg` object contains the message to be enqueued or dequeued. The message can be a `RAW` message or any user-defined type.

The following examples illustrate how to enqueue RAW messages from the DBQ queue. Note that the DBQ queue must already be created in the database.

```
Dim Q as OraAQ
Dim Msg as OraAQMsg
set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set empDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
Set Q = empDb.CreateAQ("DBQ")
Retrieve the message object from the Q object.
set Msg = Q.AQMsg
Specify the message value.
Msg.Value = "This is the first Test message"
Enqueue the message.
Q.Enqueue
```

The following lines enqueue a high priority message.

```
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Msg.Delay = 5
Msg.Value = "Urgent message"
Q.Enqueue
```

The following example dequeues the RAW messages from Oracle Database and displays the message content.

```
Q.Dequeue
MsgBox Msg.value
Dequeue and display the first high priority message
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Q.Dequeue
MsgBox Msg.value
```

See Also: OraAQ Object on page 9-3

Monitoring Messages

The OraAQ monitor methods (MonitorStart and MonitorStop) provide asynchronous dequeuing through notifications. This is suitable for applications that prefer to process messages in nonblocking mode. Applications can request to be notified on arrival of messages, by supplying an Automation object to the Monitor method. This object implements a method called NotifyMe to receive notifications. Messages can be monitored based on consumer name, message ID, or correlation.

The following sample code demonstrates a simple use of this facility. It illustrates a computerized trading system that executes buy/sell limit orders.

The sample instantiates a queue object for the STOCKS_TO_TRADE queue and monitors messages intended for consumer BROKER_AGENT. STOCKS_TO_TRADE queues messages of the user-defined type TRADEORDER_TYPE. This encapsulates all the information required to initiate a trade order. When messages addressed to the BROKER_AGENT are dequeued, the NotifyMe method of the CallbackClient object is invoked, and a stock trade is performed.

```
'First instantiate the CallbackClient. The queue monitor
' will invoke the NotifyMe on this class module.
Public CB_Client As New CallbackClient

  Dim DB As OraDatabase
  Dim Q as OraAQ
  set Q = DB.CreateAQ("STOCKS_TO_TRADE")
```

```
'Notify by calling cbclient::NotifyMe when there are messages
' for consumer '"BROKER_AGENT"
  Q.consumer = "BROKER_AGENT"

'Note that cbclient is a dispatch interface that supports the NotifyMe method.
  Dim s as string
  s = "BROKER_AGENT"
 'Notify the client only when there are messages for "BROKER_AGENT"
  Q.MonitorStart CB_Client, Q, s, 1
'other processing is performed here...

  Q.MonitorStop
Return
'Now implement the NotifyMe method of the CallbackClient class module
'and the necessary arguments that will contain the dequeued message
'NotifyMe is the callback interface defined by user. Ctx here is the
'Q object passed in at the time of MontiorStart.
Public sub NotifyMe (ByVal Ctx As Variant, ByVal Msgid As Variant )
  On Error GoTo NotifyMeErr
  Dim tradingSignal as OraAQMsg
  'Tradeorder contains details of the customer order
  Dim tradeorder as OraObject
  If IsNull(Msgid) Then
          MsgBox "No Message"
          'Get Error
          MsgBox OraDatabase.LastServerErrText
  Else
          mvarMsgid = Msgid
          Set tradingSignal = Ctx.AQMsg(1,"STOCK_TYPE","TRADER")
          set tradeorder = tradingSignal.Value

          'Tradeorder is the object of UDT "STOCK_TYPE"Access signal attribute
          'of tradeorder as tradeorder("signal).Value or tradeorder!signal
          if (tradeorder!signal = "SELL")
            'Sell the stock
             SellStock(tradeorder!NoOfShares, tradeorder!Ticker, _
                    tradeorder!Price,  tradeorder!ValidUntil)
          else if (tradeorder!signal = "BUY")
             'Buy the stock
              BuyStock(tradeorder!NoOfShares,tradeorder!Ticker, _
                   tradeorder!Price,tradeorder!ValidUntil)
          end if
  End If
NotifyMeErr:
  Call RaiseError(MyUnhandledError, "newcallback:NotifyMe Method")
End Sub
```

Database Events

Oracle Database supports detection and run-time publication of database events.

The database event publication feature allows applications to subscribe to database events just as they subscribe to messages from other applications.

Users can enable the publication of the following events:

- DML events (`DELETE`, `INSERT`, `UPDATE`)
- DDL events (`CREATE`, `ALTER`, `DROP`)

- Database events (SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN)

The event publication subsystem is integrated with the AQ publish and subscribe engine.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for a complete description of triggers for data and system events

Oracle Objects for OLE provides functionality to enable COM users to subscribe to Oracle Database events.

This feature supports asynchronous notification of database events to interested subscribers. Under this model, the client can subscribe to be notified of a database or system event, with each request stored as a subscription.

When the database event of interest fires, the subscriber is notified by the database event handler. The event handler was registered at the time of the event's subscription.

OO4O provides the OraSubscription object that represents the subscription to a database event and the OraSubscriptions collection that maintains a list of OraSubscription objects.

To subscribe to a database event, you must:

- Create a subscription, based on the database event of interest.
- Provide a database event handler. The database event handler should be an automation object that implements the NotifyDBEvents method. The NotifyDBEvents method is invoked by OO4O when the subscribed database events are fired.
- Register the subscription, using the Register method.

Example: Registering an Application for Notification of Database Events

In the following example, an application subscribes for notification of database logon events (such as all logons to the database). When a user logs on to the database, the NotifyDBEvents method of the DBEventsHdlr that was passed in at the time of subscription is invoked. The context-sensitive information and the event-specific information are passed into the NotifyDBEvents method.

The DBEventsHdlr in this example is DBEventCls, which is defined later.

The main application is as follows:

```
' First instantiate the dbevent handler. The dbevent notification
' will fire the NotifyDBEvents on the callback handler.

Public DBEventsHdlr As New DBEventCls
Private Sub Form_Load()
    Dim gOraSession As Object
    Dim gOraSubscriptions As OraSubscriptions
    Dim gOraDatabase As OraDatabase

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

   'Create the OraDatabase Object by opening a connection to Oracle.
    Set gOraDatabase = gOraSession.DbOpenDatabase
             ("ora90.us.oracle.com", "pubsub/pubsub",
              ORADB_ENLIST_FOR_CALLBACK)
```

```
    Set gOraSubscriptions = gOraDatabase.Subscriptions
    gOraSubscriptions.Add "PUBSUB.LOGON:ADMIN", DBEventsHdlr,
             gOraDatabase
    gOraSubscriptions(0).Register
    MsgBox "OK"
End Sub
```

The database event handler class that defines the `NotifyDBEvents` method is as follows:

```
Public countofMsgs as integer
Public Function NotifyDBEvents(Ctx As Variant, Payload As Variant )
    On error goto NotifyMeErr

    MsgBox "Retrieved payload " + Payload
   ' do something - here the subscription is unregistered after
   ' receiving 3 notifications
    countofMsgs = countofMsgs + 1
    If countofMsgs > 3 Then
        Ctx.Subscriptions(0).UnRegister
    End If
    Exit Sub
NotifyMeErr:
    Call RaiseError(MyUnhandledError, "newcallback:NotifyMe Method")

End Sub
```

See Also:

- OraSubscription Object on page 9-61
- OraSubscriptions Collection on page 9-70
- Register Method on page 10-229
- "Triggers on System Events and User Events" in *Oracle Database Concepts*

Application Failover Notifications

Application failover notifications can be used in the event of the failure of one database instance and failover to another instance. Because delay can occur during a failover, the application developer may want to inform the user that a failover is in progress, and request that the user stand by. Additionally, the session on the initial instance may have received some `ALTER SESSION` commands. These are not automatically replayed on the second instance. Therefore, the developer may want to replay these `ALTER SESSION` commands on the second instance.

Failover Notification Registration

To address the problems described, OO4O supports application failover notifications. To receive failover notifications, a notification handler must be registered with the `MonitorForFailover` method of the `OraDatabase` object. The notification handler must be an automation object (class module in Visual Basic) that implements the `OnFailover` method. An `IDispatch` pointer to this automation object must be passed in, along with any client-specific context, at the time of registering for failover notifications.

In the event of failover, the OnFailover method is invoked several times during the course of reestablishing the user's session. The first call to the OnFailover method of the notification handler occurs when the database first detects an instance connection loss. This is intended to allow the application to inform the user of an upcoming delay. If a failover is successful, a second call to the OnFailover method occurs when the connection is reestablished and usable. At this time, the client may want to replay the ALTER SESSION commands and inform the user that a failover has happened.

If a failover is unsuccessful, then the OnFailover method is called to inform the application that the failover will not take place.

An example of failover registration is included as part of the example in the next section.

See Also:

- MonitorForFailover Method on page 10-194
- OraDatabase Object on page 9-28
- *Oracle Net Services Administrator's Guide*for detailed information about application failover

Enabling Failover

To enable failover notifications, the option ORADB_ENLIST_FOR_CALLBACK must be passed into the call to the OpenDatabase method.

See Also: OpenDatabase Method on page 10-212

Example: Failover Notification

The following sample shows a typical developer-defined OnFailover implementation and demonstrates how to register an application.

```
'Implement the OnFailover method of the FailoverClient class module and the
' necessary arguments that will contain the dequeued message. Ctx  here is
' the application-defined context sensitive object that was passed
' in while registering with MonitorForFailover.
' An error of OO4O_FO_ERROR indicates that failover was unsuccessful, but the
' application can handle the and retry failover by returning
' a value of OO4O_FO_RETRY

Public Function OnFailover(Ctx As Variant, fo_type As Variant,fo_event _
                     as variant, fo_OraDB as Variant)
Dim str As String

OnFailover=0
str = Switch(fo_type = 1&, "NONE", fo_type = 2&, "SESSION", fo_type = _
                  4&, "SELECT")
If IsNull(str) Then
   str = "UNKNOWN!"
End If
If fo_event= OO4O_FO_ERROR Then
   MsgBox "Failover error gotten. Retrying "
   OnFailover = OO4O_FO_RETRY
   End If
If fo_event = OO4O_FO_BEGIN Then
   MsgBox " Failing Over .... with failover type : " & str
Else
   MsgBox "Failover Called with event : " & fo_event
End If
```

```
End Function
```

Registering the Application to Receive Failover Notifications

```
' First instantiate the Failover_Client. The Failover notification
' will invoke the OnFailover on this class module

Public Failover_Client As New FailoverClient
Dim OraDatabase As OraDatabase
Dim OraSession As OraSession
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

' Pass in the entire database name (ie., the entire Tnsnames entry
' with the domain name)in the opendatabase call
Set OraDatabase = OraSession.DbOpenDatabase("Exampledb.us.oracle.com", _
          "scott/tiger", ORADB_ENLIST_FOR_CALLBACK)
OraDatabase.MonitorForFailover Failover_Client, OraDatabase
```

XML Generation

Oracle Objects for OLE support for XML enables you to extract data in XML format from an Oracle database.

Data in XML markup language can be integrated with other software components that support XML. Web servers can provide XML documents along with a style sheet, thus separating the data content from its presentation, and preserving the data in its native form for easy searching.

Using Extensible Stylesheet Language Transformations (XSLT), developers can reformat XML documents received from other businesses into their desired style.

For more information about XML, go to

http://www.w3.org/XML/

XML Generation Example

OO4O renders XML from the contents of any `OraDynaset` method based on a starting row number and continuing for up to a specified amount of rows. For example:

OO4O Code

```
Dim XMLString As String
Dim startrow as Integer
Dim maxrows as Integer
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)
Set OraDynaset = OraDatabase.CreateDynaset("select EMPNO, ENAME, COMM, JOB " & _
           "from  EMP", 0&)
startrow = 4
maxrows = 2

'Output at most 2 rows beginning at row 4
XMLString = OraDynaset.GetXML(startrow, maxrows)
```

XML Output

```
<?xml version = "1.0"?>
<ROWSET>
```

```
<ROW id="4">
<EMPNO>7566</EMPNO>
<ENAME>JONES</ENAME>
<JOB>MANAGER</JOB>
</ROW>
<ROW id="5">
<EMPNO>7654</EMPNO>
<ENAME>MARTIN</ENAME>
<COMM>1400</COMM>
<JOB>SALESMAN</JOB>
</ROW>
</ROWSET>
```

The format of the XML can be customized through the `OraDynaset` and `OraField` methods:

```
Dim XMLString As String
Dim startrow as Integer
Dim maxrows as Integer
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)
Set OraDynaset = OraDatabase.CreateDynaset("select EMPNO, ENAME, COMM," & _
            "JOB from EMP", 0&)

'Change the root tag of the XML document
OraDynaset.XMLRowsetTag = "ALL_EMPLOYEES"

'Change the row tag of the XML document
OraDynaset.XMLRowTag = "EMPLOYEE"

'Remove the rowid attribute
OraDynaset.XMLRowID = ""

'Turn on the null indicator
OraDynaset.XMLNullIndicator = True

'Change the EMPNO tag name
Set EmpnoField = OraDynaset.Fields("EMPNO")
EmpnoField.XMLTagName = "EMP_ID"

'and make it an attribute rather than an element
EmpnoField.XMLAsAttribute = True

'Change the ENAME tag name
Set EnameField = OraDynaset.Fields("ENAME")
EnameField.XMLTagName = "NAME"

'Change the COMM tag name
Set CommField = OraDynaset.Fields("COMM")
CommField.XMLTagName = "COMMISSION"

'Change the JOB tag name
Set JobField = OraDynaset.Fields("JOB")
JobField.XMLTagName = "JOB_TITLE"
startrow = 4
maxrows = 2

'Output at most 2 rows beginning at row 4
XMLString = OraDynaset.GetXML(startrow, maxrows)
```

Output

```
<?xml version = "1.0"?>
<ALL_EMPLOYEES>
<EMPLOYEE EMP_ID="7566">
<NAME>JONES</NAME>
<COMMISSION NULL="TRUE"></COMMISSION>
<JOB_TITLE>MANAGER</JOB_TITLE>
</EMPLOYEE>
<EMPLOYEE EMP_ID="7654">
<NAME NULL>MARTIN</NAME>
<COMMISSION>1400</COMMISSION>
<JOB_TITLE>SALESMAN</JOB_TITLE>
</EMPLOYEE>
</ALL_EMPLOYEES>
```

See Also: OraDynaset Object on page 9-30

Datetime and Interval Data Types

From Release 9.2.0.4 and later, OO4O provides four new objects that enable developers to access and manipulate the new datetime and interval data types introduced in Oracle9*i*. Table 4–3 describes the OO4O objects and matching data types.

Table 4–3 Datetime and Interval Data Types

OO4O Objects	Oracle Data Types
`OraIntervalDS`	`INTERVAL DAY TO SECOND`
`OraIntervalYM`	`INTERVAL YEAR TO MONTH`
`OraTimeStamp`	`TIMESTAMP`
`OraTimeStamp`	`TIMESTAMP WITH LOCAL TIME ZONE`
`OraTimeStampTZ`	`TIMESTAMP WITH TIME ZONE`

Instances of these types can be fetched from the database or passed as input or output variables to SQL statements and PL/SQL blocks, including stored procedures and functions.

These new data types are not supported as elements in collections such as PL/SQL indexed tables, `VARRAY`s, or nested tables.

Obtaining Datetime and Interval Data Types

OO4O datetime and interval data types can be obtained using:

- The `Value` property of an `OraField` object in a dynaset.
- The `Value` property of an `OraParameter` object as an input or an output parameter in SQL statements or PL/SQL blocks.
- An attribute of another object or `REF`.
- The following `OraSession` methods:
  - `CreateOraIntervalDS`
  - `CreateOraIntervalYM`
  - `CreateOraTimeStamp`

- CreateOraTimeStampTZ

Descriptions of Datetime and Interval Data Types

- OraTimeStamp object

  Provides methods for operations on Oracle TIMESTAMP or TIMESTAMP WITH LOCAL TIME ZONE data types. Operations include accessing the datetime values and performing datetime operations.

- OraTimeStampTZ object

  Provides methods for operations on Oracle TIMESTAMP WITH TIME ZONE data types. Operations include accessing the datetime and time zone values and performing datetime operations.

- OraIntervalDS object

  Provides methods for operations on the Oracle INTERVAL DAY TO SECOND. This data type represents a period of time in terms of days, hours, minutes, seconds, and nanoseconds.

- OraIntervalYM object

  Provides methods for operations on the Oracle INTERVAL YEAR TO MONTH. This data type represents a period of time in terms of years and months.

Database Schema Objects

The OraMetaData interface provides access to the schema information of database objects. It is returned by invoking the Describe method of the OraDatabase interface. The Describe method takes the name of a schema object, such as the emp table and returns an OraMetaData object. The OraMetaData object provides methods for dynamically navigating and accessing all the attributes (OraMDAttribute collection) of a schema object described.

The following Visual Basic script shows a simple example of the OraMetaData interface. The sample retrieves and displays several attributes of the emp table.

```
Dim empMD as OraMetaData

set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
set empDb = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0)

'Add EMPNO as an Input parameter and set its initial value.
Set empMd = empDb.Describe("emp")

'Get the column attribute collections.
Set empColumnsMd = empMd("ColumnList").Value

'Display name, data type, and size of each column in the emp table.
For I = 0 To empColumnsMd.Count - 1
   Set ColumnMd = empColumnsMd(I).Value
   MsgBox ColumnMd("data type").Value
   MsgBox ColumnMd("Name").Value
Next I
```

See Also:

- OraMetaData Object on page 9-39
- OraMDAttribute Object on page 9-38

5

Tuning and Troubleshooting

This chapter provides information about tuning, troubleshooting, and error handing in Oracle Objects for OLE (OO4O).

This chapter contains these topics:

- Tips and Techniques for Performance Tuning
- Oracle Objects for OLE Error Handling
- Troubleshooting

Tips and Techniques for Performance Tuning

The following topics are intended to help tune the performance of applications that use Oracle Objects for OLE.

This section contains these topics:

- Early Binding of OO4O Objects
- Tuning and Customization
- Avoiding Multiple Object Reference
- Parameter Bindings
- Array Processing
- Using Read-Only, Forward-Only Dynaset
- Using the PL/SQL Bulk Collection Feature
- Migration from LONG RAW to LOB or BFILE
- Using Connection Pooling

Early Binding of OO4O Objects

The early binding technique tightly typecasts OO4O objects to their native object types rather than the generic object type provided by Visual Basic. These objects are declared directly as OO4O objects, rather than as generic objects which are later reclassified as OO4O objects. Early binding improves performance by reducing frequent access to the OO4O type library. For example:

```
'Early binding of OO4O objects
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
```

```
'Generic binding of OO4O objects

Dim OraSession as Object
Dim OraDatabase as Object
Dim OraDynaset as Object
```

To use early binding of OO4O objects, the Oracle In-Process Server type library must be referenced in the Visual Basic projects.

> **See Also:** "Using Oracle Objects for OLE Automation with Visual Basic" on page 2-2

Tuning and Customization

Data access can be tuned and customized by altering the cache and fetch parameters of a dynaset. Setting the `FetchLimit` parameter to a higher value increases the number of rows that are fetched with each request, thus reducing the number of network trips to Oracle Database, and improving performance.

The cost of increasing the size of the `FetchLimit` parameter is that it increases memory requirements on the client side, and causes more data to be swapped to and from the temporary cache file on disk. The proper `FetchLimit` value should be set according to the client computer configuration and the anticipated size of the query result.

The `FetchLimit` value can be set in the following ways:

- By using the `CreateCustomDynaset` method
- By modifying parameters of the OO4O entry in the Windows registry

For Windows, the registry key is `HKEY_LOCAL_MACHINE` and the subkey is `software\oracle\`*`KEY_HOMENAME`*`\oo4o`, where *`HOMENAME`* is the appropriate Oracle home. The OO4O installation creates the following section in the registry:

```
"FetchLimit" = 100
```

> **See Also:**
>
> - FetchLimit Property on page 11-61
> - CreateDynaset Method on page 10-85
> - OraDynaset Object on page 9-30

Avoiding Multiple Object Reference

Improper coding techniques with unnecessary object references can also affect performance. During dynaset object navigation, you should reduce the number of object references to the `OraFields` collections and `OraField` objects. The following is an inefficient code block:

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

'Traverse until EOF is reached
Do Until OraDynaset.EOF
    msgbox OraDynaset.Fields("sal").value
OraDynaset.MoveNext
Loop
```

The OraDynaset, OraFields collections, and OraField objects are referenced for each iteration. Although OO4O provides improvement in handling the field collections object, multiple references to the automation object goes though the underlying OLE/COM automation layer, which slows down the execution.

The following example shows how to reference fields through a field object and not through the fields collection of the dynaset. Testing has determined that this small amount of extra code greatly improves performance.

```
Dim flds() As OraField
Dim i, fldcount As Integer

' Create the OraDynaset Object
Set OraDynaset = OraDatabase.CreateDynaset("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 - 1
Set flds(i) = OraDynaset.Fields(i)
Next I
'Traverse until EOF is reached

Do Until OraDynaset.EOF
   msgbox Flds(5).Value
   msgbox Flds(6).Value
OraDynaset.MoveNext
Loop
```

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

Parameter Bindings

OO4O provides a way of enabling and disabling parameter object binding at the time it processes the SQL statement. This can be done through the AutoBindDisable and AutoBindEnable methods of the OraParameter object. If the SQL statement does not contain the parameter name, it is better to disable the OraParameter object because it avoids an unnecessary reference to the parameter object. This is most effective when the application is written primarily using PL/SQL procedures. For example:

```
Set OraDatabase = OraSession. OpenDatabase("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

'Add the job input parameter with initial value MANAGER.
OraDatabase.Parameters.Add "EmpCur", 0, 1
OraDatabase.Parameters("Empcur").ServerType = ORATYPE_CURSOR

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

set OraSqlStmt = CreateSQL("Begin GetEmpData(:Empcur, :deptno) End;",0&)
```

Note how the `job` parameter object is not referenced while processing the PL/SQL statement.

See Also:

- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41

Array Processing

OO4O supports an array interface to an Oracle database through the `OraParamArray` object. The array interface enables the transfer of bulk of data in single network trip. This is especially helpful while processing a PL/SQL or SQL statement through the `ExecuteSQL` or `CreateSQL` method. For example, in order to insert 100 rows into remote database without array processing, `ExecuteSQL` or `CreateSQL` must be called 100 times, which in turn makes 100 network trips. For example:

```
For I = 1 to 100
   OraParameter("EMPNO").Value = xxxx
   OraParameter("ENAME").Value = 'yyyy'
   OraParameter("DEPTNO").Value = zz
   OraDatabase.ExecuteSql("insert into emp values (:EMPNO,:ENAME,:DEPTNO)");
Next I
```

The following example makes use of arrays and makes only one network trip.

```
'ENAMEARR,:EMPNOARR,:DEPTNOARR are parameter arrays

For I = 1 to 100
   OraParameter("EMPNOARR").Put_Value xxxx, I
   OraParameter("ENAMEARR").Put_Value 'yyyy' ,I
   OraParameter("DEPTNOARR").Put_Value zz, I
Next I

'Now call the ExecuteSQL only once
OraDatabase.ExecuteSql("insert into emp values(:EMPNOARR," & _
              ":ENAMEARR, :DEPTNOARR)");
```

See Also: OraParamArray Object on page 9-47 for more information on using arrays

Using Read-Only, Forward-Only Dynaset

If your application does not make any updates to the dynaset, then you can create a read-only dynaset with the `ORADYN_READONLY` (`H4`) option. With this option, performance improvement can be gained by eliminating the overhead of parsing SQL statements locally and reducing network trips for SQL statement execution.

If your application does not need a scrollable dynaset, then you can create a forward-only dynaset with the `ORADYN_NOCACHE` (`H8`) option. With this option, performance improvement can be gained by eliminating the overhead of creating a local cache file and the overhead of reading/writing data from that file.

Using the PL/SQL Bulk Collection Feature

The PL/SQL bulk collection feature enables the selecting of bulk data in a single network trip using PL/SQL anonymous blocks. The OO4O `OraDynaset` object selects

arrays of data during SQL statement execution. This involves overhead such as performing more network round-trips, creating more cache files and internal objects. If you do not want to use a dynaset due to its overhead, then this feature is useful for selecting arrays of data. The data to be selected can be bound either as an `OraParamArray` object or as an `OraCollection` object.

The following example illustrates PL/SQL bulk collection features using the `OraCollection` interface. It shows how arrays of `enames` are selected with one network round-trip and less overload.

```
Set OraDatabase = OraSession.OpenDatabase("Exampledb", "scott/tiger", 0&)

'create a VARRAY type ENAMELIST in the database
OraDatabase.ExecuteSQL ("create type ENAMELIST as VARRAY(50) OF VARCHAR2(20)")

'create a parameter for ENAMELIST VARRAY
OraDatabase.Parameters.Add "ENAMES", Null, ORAPARM_OUTPUT, 247,"ENAMELIST"

'execute the statement to select all the enames from ename column of emp table
OraDatabase.ExecuteSQL ("BEGIN select ENAME bulk collect into" & _
                ":ENAMES from emp; END;")

'here OraParameter object returns EnameList OraCollection
Set EnameList = OraDatabase.Parameters("ENAMES").Value

'display all the selected enames
FOR I = 1 to EnameList.Size
   msgbox Enamelist(I)
NEXT I
```

See Also: OraDynaset Object on page 9-30

Migration from LONG RAW to LOB or BFILE

Oracle8*i* introduced the following new types described in "Using Large Objects (LOBs)" on page 4-3:

- `BLOB`
- `CLOB`
- `BFILE`

The design of these types allows OO4O to access them much faster than using `LONG` or `LONG RAW` types. For this reason, convert existing `LONG RAW` code to `BLOB`, `CLOB`, and `BFILE`, and only use LOBs and `BFILEs` for new applications. The OraLOB object should be used to access LOB and `BFILE` types, rather than these `LONG RAW` chunking methods, which are provided for backward compatibility only. Note that OraLOB offers maximum control.

LOB data types differ from `LONG` and `LONG RAW` data types in several ways:

- A table can contain multiple LOB columns, but can contain only one `LONG` column.
- A table containing one or more LOB columns can be partitioned, but a table containing a `LONG` column cannot be partitioned.
- The maximum size of a LOB is 4 gigabytes, but the maximum size of a `LONG` is 2 gigabytes.

- LOBs support random access to data, but LONGs data types support only sequential access.
- LOB data types (except NCLOB) can be attributes of a user-defined object type, but LONG data types cannot.
- LOB client-side buffering is used to optimize multiple small writes.
- LOB data can be stored in operating system files outside of database tablespaces (BFILE types).

To make migration easier, the following methods can be used with BLOB, CLOB, and BFILE types:

- AppendChunk Method on page 10-28
- AppendChunkByte Method on page 10-30
- GetChunk Method on page 10-156
- GetChunkByte Method on page 10-158
- GetChunkByteEx Method on page 10-160
- ReadChunk Method on page 10-224

For older applications using the LONG RAW chunking methods, migration should not require a lot of changes to the code. The primary code changes involve the requirement that null BLOB and CLOB types be updated with empty before being used.

Using Connection Pooling

The connection pool in OO4O is a pool of OraDatabase objects. An OO4O connection pool is a group of (possibly) already connected OraDatabase objects. For applications that require constant connections and disconnections to the database, such as ASP Web applications, using a connection pool results in enhanced performance.

> **See Also:** "Using the Connection Pool Management Facility" on page 3-8

Oracle Objects for OLE Error Handling

OO4O errors are grouped in the following categories:

- OLE Automation Errors
- Nonblocking Errors
- Find Method Parser Errors
- Find Method Run-Time Errors
- OraObject Instance Errors
- LOB Errors
- Oracle Streams Advanced Queuing Errors
- OraCollection Errors
- OraNumber Errors
- Oracle Errors
- Oracle Data Control Errors

OLE Automation Errors

The programmatic interface of the OO4O automation server is the OO4O In-Process Automation server. Errors that occur during execution of methods are frequently reported as an OLE Automation Error (`ERR = 440, ERROR$="OLE Automation Error"`).

When an error occurs, check the `LastServerErr` property of the `OraSession` and `OraDatabase` objects to determine whether an Oracle database error has occurred. If the `LastServerErr` is not zero, then an error has been raised by the OO4O automation server.

To find OO4O automation server errors, scan the string returned by the `ERROR$` function for the string `"OIP-NNNN"` where `NNNN` is an error number included in the Table 5–1.

Note: These values are included in the `oraconst.txt` file in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\oo4o` directory.

See Also:

- "Oracle Objects for OLE In-Process Automation Server" on page 1-2
- LastServerErr Property on page 11-87
- OraSession Object on page 9-58
- OraDatabase Object on page 9-28

Table 5–1 lists the Oracle OLE automation errors.

Table 5–1 Oracle OLE Automation Errors

Constant	Value	Description
`OERROR_ADVISEULINK`	`4096`	Internal error: Invalid advisory connection.
`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_NOFIELDINDEX`	`4100`	An invalid field index was specified. The range of indexes is `0` to `FieldCount-1`.
`OERROR_TRANSIP`	`4101`	A `BeginTrans` operation was specified while a transaction was already in progress.
`OERROR_TRANSNIPC`	`4104`	A `CommitTrans` operation was specified without first executing a `BeginTrans`.
`OERROR_TRANSNIPR`	`4105`	A `Rollback` operation was specified without first executing a `BeginTrans`.
`OERROR_NODSET`	`4106`	Internal error: System attempted to remove a nonexistent dynaset.
`OERROR_INVROWNUM`	`4108`	An attempt was made to reference an invalid row. This happens when `EOF` or `BOF` is `True`, or when the current row was deleted and no record movement occurred.
`OERROR_TEMPFILE`	`4109`	An error occurred while trying to create a temporary file for data caching.

Table 5–1 (Cont.) Oracle OLE Automation Errors

Constant	Value	Description
OERROR_DUPSESSION	4110	An attempt was made to create a named session that already exists, using the CreateSession or CreateNamedSession method.
OERROR_NOSESSION	4111	Internal error: System attempted to remove a nonexistent session.
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: Duplicate connection name.
OERROR_NOCONN	4114	Internal error: System attempted to remove a nonexistent connection.
OERROR_BFINDEX	4115	An invalid field index was specified. The range of indexes is 0 to Count -1.
OERROR_CURNREADY	4116	Internal error: System attempted to move to a row but the dynaset does not support this operation.
OERROR_NOUPDATES	4117	An attempt was made to change the data of a nonupdatable dynaset.
OERROR_NOTEDITING	4118	An attempt was made to change the value of a field without first executing the Edit method.
OERROR_DATACHANGE	4119	An attempt was made to edit data in the local cache, but the data on Oracle Database was changed.
OERROR_NOBUFMEM	4120	Out of memory for data binding buffers.
OERROR_INVBKMRK	4121	An invalid bookmark was specified.
OERROR_BNDVNOEN	4122	Internal error: Bind variable was not enabled.
OERROR_DUPPARAM	4123	An attempt was made to create a named parameter using the Add method, but that name already exists.
OERROR_INVARGVAL	4124	An invalid offset or length parameter was passed to the GetChunk method, or an internal error occurred using the AppendChunk method.
OERROR_INVFLDTYPE	4125	An attempt was made to use the GetChunk or AppendChunk method on a field that was not either Long or Long Raw type.
OERROR_INVARG	4126	An invalid argument value was entered.
OERROR_TRANSFORUP	4127	A SELECT ... FOR UPDATE operation was specified without first executing the BeginTrans operation.
OERROR_NOTUPFORUP	4128	A SELECT ... FOR UPDATE operation was specified, but the query is nonupdatable.
OERROR_TRANSLOCK	4129	A Commit or Rollback was executed while a SELECT ... FOR UPDATE operation was in progress.
OERROR_CACHEPARM	4130	An invalid cache parameter was specified. Note that the maximum value for the CacheBlocks parameter is 127.

Table 5–1 (Cont.) Oracle OLE Automation Errors

Constant	Value	Description
OERROR_FLDRQROWID	4131	An attempt was made to reference a field that requires a ROWID (Long or Long Raw), but the ROWID value was not available.
OERROR_OUTOFMEMORY	4132	Internal Error: Out of memory.
OERROR_MAXSIZE	4135	Element size specified in the AddTable method exceeds the maximum size allowed for that variable type. See "AddTable Method" on page 10-23 for more details.
OERROR_INVDIMENSION	4136	Dimension specified in the AddTable method is invalid (that is, negative). See "AddTable Method" on page 10-23 for more details.
OERROR_ARRAYSIZ	4138	Dimensions of array parameters used in the INSERT, UPDATE, and DELETE statements are not equal.
OERROR_ARRAYFAILP	4139	Error processing arrays. For details see the oo4oerr.log in the Windows directory.
OE_CLIPFAIL	4141	Internal error: Clipboard could not be opened or closed.
OE_NOSOURCE	4143	No source string was provided for the UpdateResource method.
OE_INVSOURCE	4144	Invalid source type was provided for UpdateSource method.
OE_PLSQLDYN	4145	An attempt was made to set SQL property for dynaset created from PL/SQL cursor.
OERROR_CREATEPOOL	4147	Database pool already exists for this session.
OERROR_GETDB	4148	Unable to obtain a free database object from the pool.
OE_INVINPUTTYP	4149	Input type is not compatible with the field or parameter type.
OE_NOEDITONCLONE	4150	An attempt was made to edit a cloned object.
OE_BNDCHGTYPERR	4152	An attempt was made to change the type of a parameter array or an array of extended type.

Nonblocking Errors

Table 5–2 lists the nonblocking errors.

Table 5–2 Nonblocking Errors

Constant	Value	Description
OERROR_NONBLKINPROGRESS	4153	Nonblocking operation in progress.
OERROR_NONONBLKINPROGRESS	4154	Operation is valid only when nonblocking operation is in progress.

Find Method Parser Errors

Find method parser errors occur when the parser cannot evaluate the expression in the Find method. These errors specify the part of the expression that caused the error.

Table 5–3 lists the Find method parser errors.

Table 5–3 Find Method Parser Errors

Constant	Value	Description
OERROR_STACK_OVER	4496	Stack overflow.
OERROR_SYNTAX	4497	Syntax error.
OERROR_MISPLACED_ PAREN	4498	Misplaced parenthesis.
OERROR_MISPLACED_ QUOTE	4499	Misplaced quotation marks.
OERROR_MISSING PAREN	4500	Warning: Missing closing parenthesis.
OERROR_EXPECTED_PAREN	4501	Open parenthesis expected.
OERROR_PARSER_UNKNOWN	4502	Unknown parser error condition.
OERROR_INVALID_ FUNCTION	4503	Syntax not supported.
OERROR_INVALID_COLUMN	4504	Invalid column name.
OERROR_MAX_TOKEN	4505	Maximum size exceeded in token.
OERROR_PARSER_DATA_ TYPE	4506	Unsupported data type.
OERROR_UNEXPECTED_ TOKEN	4507	Unexpected token found.
OERROR_END_OF_CLAUSE	4508	Unexpected end of clause.

Find Method Run-Time Errors

Find method run-time errors occur when the system cannot evaluate a find expression. Such errors are rare. When one occurs, the parser could have generated incorrect code.

Table 5–4 lists the Find method run-time errors.

Table 5–4 Find Method Run-Time Errors

Constant	Value	Description
OERROR_INVALID_INSTR	4516	Internal error: Invalid instruction.
OERROR_STACK_ERROR	4517	Internal error: Stack overflow or underflow.
OERROR_CONVERT_TYPES	4518	Invalid type conversion.
OERROR_RUNTIME_DATA_TYPE	4519	Invalid data type.
OERROR_INVALID_SQL_ARG	4520	SQL function missing an argument.
OERROR_INVALID_COMPARE	4521	Invalid comparison.
OERROR_SELECT_DUAL	4522	SELECT from dual failed.
OERROR_DUAL_DATATYPE	4523	Invalid data type in SELECT from dual.
OER_OPER	4524	Invalid use of operator.

OraObject Instance Errors

Table 5–5 lists the OraObject instance errors.

Table 5–5 OraObject Instance Errors

Constant	Value	Description
OERROR_NOOBJECT	4796	Creating an OraObject object instance in the client-side object cache failed.
OERROR_BINDERR	4797	Binding an OraObject object instance to a SQL statement failed.
OERROR_NOATTRNAME	4798	Getting the attribute name of an OraObject object instance failed.
OERROR_NOATTRINDEX	4799	Getting the attribute index of an OraObject object instance failed.
OERROR_INVINPOBJECT	4801	Invalid input object type for the binding operation.
OERROR_BAD_INDICATOR	4802	Fetched OraObject instance has an invalid indicator structure.
OERROR_OBJINSTNULL	4803	Operation on the NULL OraObject instance failed. See the "IsNull (OraObject) Property" on page 11-81.
OERROR_REFNULL	4804	Pin operation on the NULL Ref value failed. See the "IsRefNull (OraRef) Property" on page 11-84.

See Also:

- IsNull (OraObject) Property on page 11-81
- IsRefNull (OraRef) Property on page 11-84
- OraObject Object on page 9-43
- OraRef Object on page 9-52

LOB Errors

Table 5–6 lists the LOB errors.

Table 5–6 LOB Errors

Constant	Value	Description
OERROR_INVSEEKPARAMS	4897	Invalid seek value is specified for the LOB read/write operation.
OERROR_LOBREAD	4898	Read operation failed.
OERROR_LOBWRITE	4899	Write operation failed.
OEL_INVCLOBBUF	4900	Input buffer type for CLOB write operation is not string.
OEL_INVBLOBBUF	4901	Input buffer type for BLOB write operation is not byte.
OERROR_INVLOBLEN	4902	Invalid buffer length for the LOB write operation.
OERROR_NOEDIT	4903	Write, Trim, Append, Copy operations are not allowed in this mode.
OERROR_INVINPUTLOB	4904	Invalid input LOB for the bind operation.
OERROR_NOEDITONCLONE	4905	Write, Trim, Append, Copy operations are not allowed for a cloned LOB object.

Table 5–6 (Cont.) LOB Errors

Constant	Value	Description
OERROR_LOBFILEOPEN	4906	Specified file could not be opened during a LOB operation.
OERROR_LOBFILEIOERR	4907	File Read or Write operation failed during a LOB operation.
OERROR_LOBNULL	4908	Operation on NULL LOB failed. See "IsNull (OraLOB/BFILE) Property" on page 11-80.

See Also:

- OraBLOB, OraCLOB Objects on page 9-11
- IsNull (OraLOB/BFILE) Property on page 11-80

Oracle Streams Advanced Queuing Errors

Table 5–7 lists the Oracle Streams Advanced Queuing errors.

Table 5–7 Oracle Streams Advanced Queuing Errors

Constant	Value	Description
OERROR_AQCREATEERR	4996	Error creating the OraAQ Object.
OERROR_MSGCREATEERR	4997	Error creating the AQMsg object.
OERROR_PAYLOADCREATEERR	4998	Error creating the payload object.
OERROR_MAXAGENTS	4999	Maximum number of subscribers exceeded.
OERROR_AGENTCREATEERR	5000	Error creating the AQAgent object.

See Also: OraAQ Object on page 9-3

OraCollection Errors

Table 5–8 lists the OraCollection errors.

Table 5–8 OraCollection Errors

Constant	Value	Description
OERROR_COLLINSTNULL	5196	Operation on NULL OraCollection failed. See "IsNull (OraCollection) Property" on page 11-79.
OERROR_NOELEMENT	5197	Element does not exist for the given index.
OERROR_INVINDEX	5198	Invalid collection index is specified.
OERROR_NODELETE	5199	Delete operation is not supported for the VARRAY collection type.
OERROR_SAFEARRINVELEM	5200	Variant SafeArray cannot be created from the collection having nonscalar element types.

See Also:

- OraCollection Object on page 9-19
- IsNull (OraCollection) Property on page 11-79

OraNumber Errors

Table 5–9 lists the `OraNumber` errors.

Table 5–9 OraNumber Errors

Constant	Value	Description
`OERROR_NULLNUMBER`	`5296`	Operation on `NULL OraNumber` object failed.

See Also: OraNumber Object on page 9-41

Oracle Errors

The most recent Oracle error text is available from the `LastServerErr` and `LastServerErrText` properties of the `OraSession` or `OraDatabase` objects.

- `OraSession` object

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

- `OraDatabase object`

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

  See Also:

  - LastServerErr Property on page 11-87
  - LastServerErrText Property on page 11-90
  - OraSession Object on page 9-58
  - OraDatabase Object on page 9-28

Oracle Data Control Errors

Oracle Data Control errors are specific to the Oracle data control. During the visual access of the data control, the OO4O automation server-specific errors are reported as OLE automation server errors with the error code of `ODCERR_AUTOMATION`. Specific Oracle Data Control error codes are retrieved from the `DataErr` parameter of the `Error()` event.

Table 5–10 lists the Oracle Data Control errors.

Table 5–10 Oracle Data Control Errors

Constant	Value	Description
`ODCERR_INITOIP`	`28000`	Initialization of Oracle In-Process Server failed. Check the registry for the correct location of Oracle In-Process Server.

Table 5–10 (Cont.) Oracle Data Control Errors

Constant	Value	Description
ODCERR_OLEQE	28001	Internal error. Querying In-Process Server interface failed.
ODCERR_AUTOMATION	28003	Oracle In-Process Server error occurred.
ODCERR_NODYNASET	28007	Attempted to access Oracle Data Control before initialization.
ODCERR_FIELDINDEX	28009	Bound controls trying to access with invalid field index.
ODCERR_FIELDNAME	28013	Bound controls tried to access with an invalid field name.
ODCERR_MEMORY	28014	Internal error. Failed to allocate memory for the requested bindings from the bound control.
ODCERR_BMKTYPE	28015	Oracle Data Control does not support the requested bookmark type.
ODCERR_CONVERSION	28016	Oracle Data Control cannot convert the field value to the requested type.
ODCERR_SETSESSION	28017	Setting the session property is not allowed.
ODCERR_SETDATABASE	28018	Setting the database property is not allowed.
ODCERR_BLOBUPDATE	28019	Oracle Data Control does not update picture or raw data directly from the bound control. Use `AppendChunk()` method.
ODCERR_DYN_NOCACHE	28020	`Recordset` property cannot be set to a dynaset created with the `ORADYN_NOCACHE` option (bound control connected to data control often requires bidirectional navigation).
ODCERR_DYN_NOMOVEFIRST	28021	`Recordset` property cannot be set to a dynaset created with the `ORADYN_NOMOVEFIRST` option.

See Also: AppendChunk Method on page 10-28

Troubleshooting

This topic describes common errors related to the following:

- OLE Initialization or OLE Automation Errors
- Oracle Network Errors
- Access Violations

See Also: *Oracle Database Error Messages* for additional information about errors

OLE Initialization or OLE Automation Errors

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

Table 5–11 lists the causes and solutions for OLE errors.

Table 5–11 Causes and Solutions for OLE Errors

Possible Cause	Solution
Your system does not contain the Microsoft OLE Automation or run-time, files or these files are out of date.	Make sure you have the latest versions of files such as the following installed. ■ `mfc42.dll` ■ `oleaut32.dll` ■ `ole32.dll`
The Oracle Objects for OLE object information was not registered in the Windows registration database.	Either reinstall Oracle Objects for OLE or run the `regedt32.exe` file to register information. See "Oracle Objects for OLE Redistributable Files" on page 1-6.
Your system does not contain the Oracle Required Support Files: ■ `oraclient*.dll` ■ `orageneric*.dll` ■ `oracommon*.dll` ■ `oracore*.dll` ■ `oranls*.dll`	Check the OO4O readme.htm file to see what version of the Oracle Database client is required and install it.
Your system does not contain the Oracle networking product or its files are not on the PATH.	Install an Oracle networking product, or add to your PATH an environment variable that indicates the directory containing these files.
You misspelled a method or property name.	Check *Oracle Objects for OLE Developer's Guide* (this guide) to determine the correct spelling.
You referenced a method or property from the wrong object.	Check *Oracle Objects for OLE Developer's Guide* (this guide) to determine the correct object.
Your system does not contain the `oraansi`*`VER`*`.dll` file.	Reinstall Oracle Objects for OLE or add to your `PATH` environment variable the directory in which these files are located. Note: *`VER`* refers to the version.

See Also: "Oracle Objects for OLE Redistributable Files" on page 1-6

Oracle Network Errors

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

Table 5–12 lists the Oracle network errors.

Table 5–12 Oracle Networking Errors

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 the Oracle networking product.	Install Oracle networking software.

See Also:

- Connect Property on page 11-23
- DatabaseName Property on page 11-37
- OpenDatabase Method on page 10-212

Access Violations

The most frequent cause of access violations is installing Oracle Objects for OLE while other applications are running that require the OO4O automation server, Oracle Required Support Files, or OLE. To avoid this, install Oracle Objects for OLE immediately after starting Windows and before running any other application.

Table 5–13 lists the access violations.

Table 5–13 Access Violations

Possible Cause	Solution
Duplicate Oracle Objects for OLE files exist in `SYSTEM` directories or along the `PATH`.	Remove any duplicate files. The files oip*`VER`*`.dll` and oip*`VER`*`.tlb` should only be located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\bin` directory.
Duplicate Oracle Required Support Files DLLs exist in the `SYSTEM` directories or along the `PATH`.	Remove any duplicate files. Typically, the Oracle Required Support Files DLLs are located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*` \bin` directory: ▪ `oraclient*.dll` ▪ `orageneric*.dll` ▪ `oracommon*.dll` ▪ `oracore*.dll` ▪ `oranls*.dll`
Duplicate OLE DLLs exist in the `SYSTEM` directories or along the `PATH`.	Remove any duplicate files. The OLE DLLs (listed in the OO4O File Locations section) should only be located in `\system` directories.

See Also: "Oracle Objects for OLE File Locations" on page 1-6

6

Quick Tour with Visual Basic

This quick tour is designed to get you started with Oracle Objects for OLE for Visual Basic. An example application, the employee database application, demonstrates how to program basic database operations, such as navigating through data and, adding, modifying, and querying records. A more advanced section demonstrates how to perform batch inserts using parameter arrays and SQL statement objects. This quick tour and example application assume that the `Scott/Tiger` schema is installed.

See Also: "Demonstration Schema and Code Examples" on page 2-1

The entire code for this example application is provided in the *ORACLE_BASE*`\`*ORACLE_HOME*`\OO4O\VB\SAMPLES\QT\` directory.

This quick tour covers the following topics:

Introduction

This section introduces the employee database application and the two Visual Basic forms that users interact with to use the application.

About the Employee Database Application

The employee database application lets the user do the following:

- Browse through data
- Add records
- Update records
- Query the database
- Add records in a batch operation

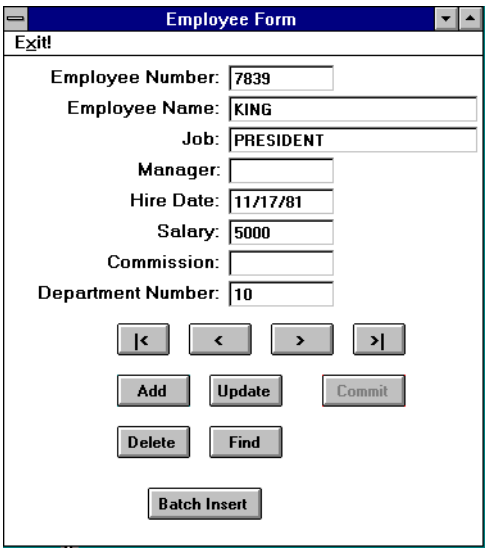
To provide these functions, this example uses the following forms:

Employee Form

The Employee Form displays the fields of the database `EMP` table and has functional buttons that allow the user to browse, add, update, and query records.

Figure 6–1 shows the Employee Form.

Figure 6–1 Employee Form



See Also:

- "Completed Sample Form_Load Procedure" on page 6-5 for the code for the `Form_Load` procedure that initializes the Employee Form
- "Programming a Data Entry Form" on page 6-6 for a detailed description of the Employee Form and code for the navigational buttons

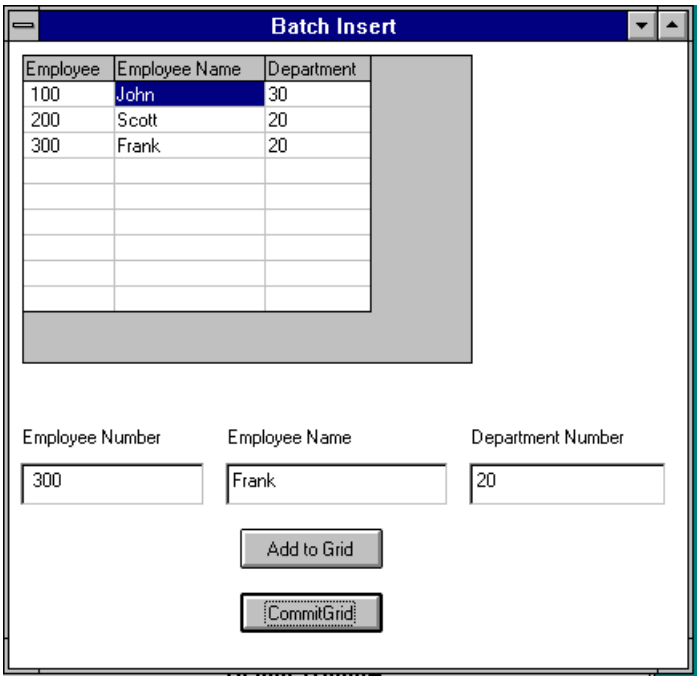
Batch Insert Form

The Batch Insert Form allows users to enter records in a batch operation.

See Also: "Programming a Batch Form" on page 6-16 for a detailed description of the Batch Insert Form and code for its commands

Figure 6–2 shows the Batch Insert Form.

Figure 6–2 Batch insert Form



Getting Started: Steps to Accessing Oracle Data

Before server data can be manipulated, the application must accomplish the four steps that are described in this section. Sample code for this example is provided in "Completed Sample Form_Load Procedure" on page 6-5.

1. Start the Oracle In-Process Automation Server.

   The Oracle In-Process Server (OIP) provides the interface between the Visual Basic application and Oracle Database. To start the Oracle In-Process Server, you must create an `OraSession` object using the Visual Basic `CreateObject()` function, as follows:

   ```
   Set OraSession = CreateObject("OracleInProcServer.XOraSession")
   ```

   When creating the `OraSession` object, the argument supplied to the `CreateObject()` function must always be `OracleInProcServer.XOraSession`. The left side of the argument defines the application name as registered in your system, in this case, `OracleInProcServer`. The right side identifies the type of object to create, in this case, the `XOraSession` object. Executing this command starts the Oracle In-Process Server.

2. Connect to Oracle Database.

   After the OIP server is running, you can connect to a local or remote Oracle database. To do so, you must create the `OraDatabase` object as follows:

   ```
   Set OraDatabase = OraSession.OpenDatabase("Exampledb", "scott/tiger", _
         ORADB_DEFAULT)
   ```

   The `OraSession.OpenDatabase()` method creates the `OraDatabase` object. The method call must specify the database name, the connection string, and a bit flag that represents the database mode. The constant `ORADB_DEFAULT` represents

the default database mode. When Visual Basic executes this line, a connection is created to the specified database.

3. Create a global `OraDynaset` object to manipulate the data.

   Oracle Objects for OLE lets users browse and update data using an object called a dynaset.

   The Employee application needs a global dynaset that the rest of the program can access. The `OraDatabase.CreateDynaset()` method creates the dynaset specifying a valid SQL `SELECT` statement. In the example, the statement selects all the rows from the `emp` table and assigns the resulting dynaset to the global `EmpDynaset` variable as follows:

   ```
   Set EmpDynaset = OraDatabase.CreateDynaset("select * from emp", _
        ORADYN_DEFAULT)
   ```

   The `CreateDynaset()` method returns a pointer to the result of the SQL SELECT statement.

   The `ORADYN_DEFAULT` parameter value specifies the default dynaset state. In the default state, Oracle Objects for OLE sets unset fields to `NULL` while adding records using the `AddNew` method. This behavior is preferable because the `emp` table has no column defaults defined. You can also specify other options to allow server column defaults when adding records.

   See Also: "CreateDynaset Method" on page 10-85

4. Refresh the Employee Form with dynaset data.

   The Employee Form displays database records one row at a time. Changes to the current row, such as those caused by navigating to a different row, must be reflected on the screen. The `EmpRefresh()` subroutine updates fields with the current dynaset row. For `NULL` field values, empty strings are displayed.

   The following is an example of an `EmpRefresh()` subroutine:

   ```
    Private Sub EmpRefresh()
   'check if the current dynaset row is valid
   If EmpDynaset.BOF <> True And EmpDynaset.EOF <> True Then

    txtEmpno = EmpDynaset.Fields("empno").Value

    ' we can't display nulls, so display ""  for NULL fields
    If Not IsNull(EmpDynaset.Fields("ename").Value) Then
      txtEname = EmpDynaset.Fields("ename").Value
      Else
        txtEname = ""
      End If

    If Not IsNull(EmpDynaset.Fields("job").Value) Then
      txtJob = EmpDynaset.Fields("job").Value
      Else
        txtJob = ""
      End If

    'check if mgr=nul
    If Not IsNull(EmpDynaset.Fields("mgr").Value) Then
     txtMgr = EmpDynaset.Fields("mgr").Value
     Else
       txtMgr = ""
     End If
   ```

```
  If Not IsNull(EmpDynaset.Fields("hiredate").Value) Then
    txtHireDate = EmpDynaset.Fields("hiredate").Value
  Else
    txtHireDate = ""
  End If

  If Not IsNull(EmpDynaset.Fields("hiredate").Value) Then
    txtSal = EmpDynaset.Fields("sal").Value
  Else
    txtSal = ""
   End If

  'check if comm=nul
  If Not IsNull(EmpDynaset.Fields("comm").Value) Then
   txtComm = EmpDynaset.Fields("comm").Value
  Else
   txtComm = ""
  End If

  txtDeptno = EmpDynaset.Fields("deptno").Value

'if the current dynaset row is invalid, display nothing
 Else

  txtEmpno = ""
  txtEname = ""
  txtJob = ""
  txtMgr = ""
  txtHireDate = ""
  txtSal = ""
  txtComm = ""
  txtDeptno = ""

 End If

End Sub
```

Completed Sample Form_Load Procedure

In the employee application described in the previous section, the `Form_Load()` procedure creates the OIP server, connects to the database, creates a global dynaset, and calls the `EmpRefresh` function to display the field values on the Employee Form. The following is an example of a `Form_Load()` procedure:

```
Private Sub Form_Load()
'OraSession and OraDatabase are global
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")
 Set OraDatabase = OraSession.OpenDatabase("Exampledb", "scott/tiger", 0&)
 Set EmpDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)

 Call EmpRefresh

End Sub
```

The following variables must be defined globally in `EMP_QT.BAS`:

```
Global OraSession As Object
Global OraDatabase As Object
```

```
Global EmpDynaset As Object
```

Programming a Data Entry Form

This section describes the Employee Form in detail and then describes the functions that it uses.

About the Employee Form

The Employee form displays the fields of the database EMP table and has functional buttons that allow the user to browse, add, update, and query records.

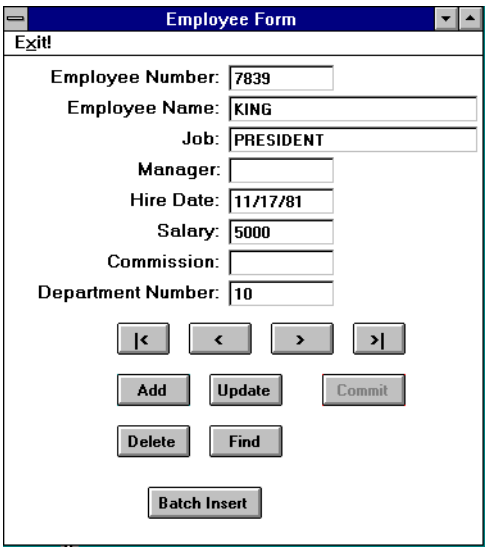
Each field corresponds to a column in the database EMP table. The Employee field (ENAME) is the indexed column and is mandatory for each record. The field data types and sizes are defined as follows in the EMP table:

```
Name                      Null?       Type
------------------------   ----------  --------------------------
EMPNO                      NOT NULL    NUMBER(4)
ENAME                                  VARCHAR2(10)
JOB                                    VARCHAR2(9)
MGR                                    NUMBER(4)
HIREDATE                               DATE
SAL                                    NUMBER(7,2)
COMM                                   NUMBER(7,2)
DEPTNO                     NOT NULL    NUMBER(2)
```

The Employee Number (EMPNO) and Department (DEPTNO) columns are NOT NULL, and, therefore, always require a value when a record is added. The length of each field is enforced by setting the MaxLength property of each TextBox to the appropriate number.

Figure 6–3 shows the Employee Form.

Figure 6–3 Employee Form



The initial code for the actual Form_Load procedure is provided in "Completed Sample Form_Load Procedure" on page 6-5.

The Employee form is initialized by the `Form_Load()` procedure and includes the following features:

- Navigating Through Data
- Adding Records
- Updating Records
- Deleting Records
- Querying the Database

Navigating Through Data

Database applications typically require that the user be able to view data in the database. The Employee form has four buttons that let the user scroll through data. Table 6–1 lists the buttons, what they do, which dynaset move method enables the action of the button, and where to look for further information.

Table 6–1 Navigational Buttons and Dynaset Move Methods

Button	Action	Method	See...
|<	Moves to the first record	`MoveFirst`	Moving to First or Last Rows
<	Moves to the previous record	`MovePrevious`	Moving to the Previous Row
>	Moves to the next record	`MoveNext`	Moving to the Next Row
>|	Moves to the last record	`MoveLast`	Moving to First or Last Rows

To enable navigation through the records of the Employee database, you must first create a global dynaset that selects all the records (rows). Then use the dynaset move methods to program the navigation buttons.

Moving to First or Last Rows

To enable a move to the first row of a dynaset, use the `MoveFirst` method. Then call the `EmpRefresh()` routine to refresh the data in the Employee form.

The following example code shows the first-click event procedure for the employee example:

```
Private Sub cmdFirst_Click()

 EmpDynaset.MoveFirst
 Call EmpRefresh

End Sub
```

For a move to the last row, use the `MoveLast` method. Then, call the `EmpRefresh()` routine to refresh the data in the Employee form.

The following example code shows the last-click event procedure for the employee example:

```
Private Sub cmdLast_Click()

 EmpDynaset.MoveLast
 Call EmpRefresh

End Sub
```

Moving to the Previous Row

Navigation is possible to any row of a dynaset. If a user is positioned in the middle of a dynaset (that is, the current row is not the first row), the `MovePrevious` method enables navigation to the previous row.

However, when a user is positioned on the first row (current row is the first row) and executes the `MovePrevious` method, the beginning-of-file (BOF) condition becomes `TRUE` and the current row becomes invalid. In this case, the current row must be reset to the first row using the `MoveFirst` method.

The following example code shows the click-event procedure for the Previous button:

```
Private Sub cmdPrevious_Click()

 If EmpDynaset.BOF <> True Then
  EmpDynaset.DbMovePrevious
  If EmpDynaset.BOF = True Then
   MsgBox WarnFirstEmp$
   EmpDynaset.DbMoveFirst
  End If
 End If
```

Moving to the Next Row

If a user is positioned in the middle of a dynaset (that is, the current row is not the last row), the `MoveNext` method enables navigation to the next row.

However, when a user is positioned on the last row (current row is the last row) and then executes `MoveNext`, the end-of-file condition (EOF) becomes `TRUE` and the current row becomes invalid. In this case, the current row must be reset to the last row using the `MoveLast` method.

The following example code shows the click-event procedure for the Next button:

```
Private Sub cmdNext_Click()

 If EmpDynaset.EOF <> True Then
  EmpDynaset.DbMoveNext
  If EmpDynaset.EOF = True Then
   MsgBox WarnLastEmp$
   EmpDynaset.DbMoveLast
  End If
 End If
```

Adding Records

In the example application, the following buttons allow users to add employee records to the database:

- Add
- Commit

To add a record, the user clicks on the Add button, enters the new fields in the text boxes, and then clicks the Commit button to save the data to the database.

Coding the Add Button

The Add event procedure must perform the following steps:

1. Clear the fields on the form.
2. Disable the Add button.
3. Enable the Commit button.
4. Let the user enter new field values.

The following example code shows the Add event procedure for the Add button:

```
Private Sub AddNew_Click()
'Blank out the fields
  txtEmpno = ""
  txtEname = ""
  txtJob = ""
  txtMgr = ""
  txtHireDate = ""
  txtSal = ""
  txtComm = ""
  txtDeptno = ""

'Disable the Add button and enable the commit button
  AddNew.Enabled = False
  Commit.Enabled = True
'Disable the navigation buttons
  DisableNavButtons
'Set doadd to true for commit procedure
  DoAdd = True
End Sub
```

When the `AddNew_Click()` method exits, control returns to the Employee Form where the user enters values in the fields.

Coding the Commit Button (Add)

To commit an addition, you must place the dynaset in add mode using the `AddNew` method. Then, you assign the new data to the dynaset fields and update the database using the `Update` method. To make the program robust, the software validates some fields before adding them to the database.

The `Commit_Click()` event procedure for adding records must do the following:

1. Check that the Employee Number and Department fields are not null.
2. Check that the new Employee Number is not a duplicate entry.

   Steps 1 and 2 are performed by the `DoValidationChecks()` function which is described following the `Commit_Click()`.
3. Place the dynaset in add mode using the `AddNew` method.
4. Assign entered data to dynaset fields using the `Fields().Value` property. This step is performed by the `UpdateDynasetFields` function.
5. Update the database with new records, using the `Update` method.
6. Disable the Commit button.
7. Enable the Add button.

The code for the `Commit` function is broken into the following routines:

- "Commit_Click Event Procedure (Add)" on page 6-10
- "DoValidationChecks( ) Function" on page 6-10

- "UpdateDynasetFields( ) Function" on page 6-11

Commit_Click Event Procedure (Add) The following is a typical `Commit_Click()` event procedure for adding records:

```
Private Sub Commit_Click()

On Error GoTo err_commit

ErrMsg = ""
'Do validation checks on entered data
If DoValidationChecks Then 'If validation checks have passed

'Add the new record to dynaset
EmpDynaset.AddNew

'Update the dynaset fields and then update database if there is no error.
If UpdateDynasetFields Then

'Update the database
  EmpDynaset.Update

Commit.Enabled = False
AddNew.Enabled = True

Exit Sub

err_commit:
    If ErrMsg <> "" Then
        MsgBox ErrMsg
    Else
        MsgBox Error$
    End If

End Sub
```

DoValidationChecks( ) Function To check for duplicate entries as suggested in Step 2, you must create a local dynaset with the `NOCACHE` option, using a SQL statement that counts the rows matching the entered Employee Number field. If a match is found (row count greater than 0), the entered employee number is a duplicate entry and an error is displayed. In this case, because the SQL `SELECT` statement returns only a number, creating the dynaset without a cache is a more efficient error check than the server finding a duplicate entery.

`DoValidationChecks()` returns `True` if the entered data is valid; otherwise, it returns `False`.

```
Function DoValidationChecks() As Boolean

Dim DupDyn As Object
Dim DupDynQry As String

On Error GoTo err_ValidationCheck

ErrMsg = ""
'Empno cannot be changed while in Update mode, so we can skip over validation
If DoAdd Then
    If txtEmpno = "" Then
        ErrMsg = "You must enter a value for Employee Number"
```

```
        Error 1
    End If
End If

If txtHireDate <> "" And Not IsDate(txtHireDate) Then
    ErrMsg = "Enter date as dd-mmm-yy."
    Error 2
End If

If txtDeptno = "" Then
    ErrMsg = "You must enter a value for Department Number"
    Error 3
End If

'If adding a record, check for Duplicate empno value by
'attempting to count rows with same value
'Build Query:
If DoAdd Then
    DupDynQry = "select count(*) from emp where empno = " & txtEmpno
    Set DupDyn = OraDatabase.CreateDynaset(DupDynQry, ORADYN_NOCACHE)

    If DupDyn.Fields(0).Value <> 0 Then
        ErrNum = DUPLICATE_KEY
        ErrMsg = "Employee Number already exists."
        Error ErrNum
    End If
End If
'Succesful validation with no errors returns True
DoValidationChecks = True
Exit Function

err_ValidationCheck:
    If ErrMsg <> "" Then
        MsgBox ErrMsg
    Else
        MsgBox Error$
    End If
    'Validation returns false on failure
    DoValidationChecks = False

End Function
```

UpdateDynasetFields( ) Function The commit event procedure calls this function after putting the dynaset in either `Edit` or `AddNew` mode. The `UpdateDynasetFields()` function sets the dynaset fields to the values entered in the text boxes. The function returns `TRUE` if successful, or returns `FALSE` if there is an error.

```
 Function UpdateDynasetFields() As Integer
'This function sets the dynaset field value to those entered in the text boxes.
'The function returns true on success, false on error.

ErrMsg = ""

On Error GoTo err_updatedynasetfields

EmpDynaset.Fields("empno").Value = txtEmpno
EmpDynaset.Fields("ename").Value = txtEname
EmpDynaset.Fields("job").Value = txtJob
EmpDynaset.Fields("mgr").Value = txtManager
```

```
EmpDynaset.Fields("hiredate").Value = txtHireDate
EmpDynaset.Fields("sal").Value = txtSal
EmpDynaset.Fields("comm").Value = txtComm
EmpDynaset.Fields("deptno").Value = txtDeptno

UpdateDynasetFields = True

Exit Function

err_updatedynasetfields:
    If ErrMsg <> "" Then
        MsgBox ErrMsg
    Else
        MsgBox Error$
    End If
    UpdateDynasetFields = False
```

Updating Records

To allow users to update existing records in the database, you need to include an Update button in the Employee Form. Users navigate to a particular record, click the Update button, make changes, and then click the Commit button.

While in update mode, the application makes the following restrictions:

- Users cannot navigate to another record or perform another function.
- Users cannot change the employee number because this is the primary key.

To program the Update function, write an event procedure for the Update button and modify the Commit procedure so that it handles both updating and adding records.

Coding the Update Button

To code the Update button, disable the Employee Number text box to prevent changes to this field while updating records, because this is a primary key. You must also disable the other buttons to disable other functions, such as navigation, while updating records.

Set the `DoUpdate` Boolean expression to `TRUE`, so the commit procedure recognizes the current process as an update operation, not an addition.

The update event procedure must do the following:

1. Disable the Update button.
2. Enable the Commit button.
3. Disable other buttons to disable functions, such as navigation, during the update operation.
4. Disable the Employee Number text box.
5. Set the `DoUpdate` flag to `True`.
6. Let the user enter changes.

The following example code shows the update event procedure:

```
Private Sub cmdUpdate_Click()
'Disable the Update button and enable the commit button
  cmdUpdate.Enabled = False
  Commit.Enabled = True
'Disable all other buttons
```

```
  DisableNavButtons

  txtEmpno.Enabled = False
  DoUpdate = True
End Sub
```

The update and add event procedures call the `DisableNavButtons()` subroutine to disable navigation and other functions during an add or update operation.

```
Private Sub DisableNavButtons()
'disable all buttons while adding and updating
cmdFirst.Enabled = False
cmdPrevious.Enabled = False
cmdNext.Enabled = False
cmdLast.Enabled = False
cmdFind.Enabled = False
cmdUpdate.Enabled = False
AddNew.Enabled = False

End Sub
```

Coding the Commit Button to Add and Update Records

The procedure for committing an update operation is similar to committing an add, except that the dynaset is set in edit mode using the `Edit` method and then the new dynaset values are assigned.

Because the same commit button and the same commit event procedure are used to add and update, two global flags `DoAdd` and `DoUpdate` are added to distinguish between adding and updating. The Add and Update click event procedures set these flags.

The Commit event procedure for adding and updating must do the following:

1. Validate entered data using the `DoValidationChecks()` function as before.
2. Use `AddNew` to add records or else use `Edit` for updates.
3. Assign entered data to dynaset fields, using the `Fields().Value` property using `UpdateDynasetFields()` as before.
4. Update database with new records, using `Update`.
5. Disable the Commit button.
6. Reenable all other functional buttons including the Add and Update buttons.
7. Set the `DoUpdate` and `DoAdd` flags to `False`.

The code that changes button and flag states in Steps 5 through 7 is provided in a new subroutine called `SetAfterCommitFlags()`. This replaces the lines of code that originally enabled `Commit` and `AddNew`.

The code for this Commit function is broken into the following routines:

- "Commit_Click( ) Event Procedure Example" on page 6-14
- "DoValidationChecks( ) Function" on page 6-10, also used in the original `Commit` function
- "UpdateDynasetFields( ) Function" on page 6-11, also used in the original `Commit` function

- "SetAfterCommitFlags() Subroutine Example" on page 6-14, which is a new subroutine

Commit_Click( ) Event Procedure Example

The following example shows the `Commit_Click` Event Procedure.

```
Private Sub Commit_Click()

On Error GoTo err_commit

ErrMsg = ""
'Do validation checks on entered data
If DoValidationChecks Then 'If validation checks have passed

    'If we are adding a record use AddNew
    If DoAdd = True Then
        EmpDynaset.AddNew
    End If
    'If we are updating a record use Edit
    If DoUpdate = True Then
       EmpDynaset.Edit
    End If
    'Update the dynaset fields and then update database if there is no error.
    If UpdateDynasetFields Then
        EmpDynaset.Update
    End If

    SetAfterCommitFlags

End If 'Endif for DoValidationChecks

Exit Sub

err_commit:
    If ErrMsg <> "" Then
        MsgBox ErrMsg
    Else
        MsgBox Error$
    End If

End Sub
```

SetAfterCommitFlags() Subroutine Example

The following example shows the `SetAfterCommitFlag()` Subroutine.

The `SetAfterCommitFlags()` subroutine is called at the end of the commit event procedure. The `SetAfterCommitFlags()` subroutine reenables disabled buttons and text boxes and sets the `DoUpdate` and `DoAdd` flags to `False`.

```
Sub SetAfterCommitFlags()
'disable commit and re-enable add and update buttons
Commit.Enabled = False
AddNew.Enabled = True
cmdUpdate.Enabled = True

'enable the other buttons
cmdFirst.Enabled = True
cmdPrevious.Enabled = True
cmdNext.Enabled = True
```

```
cmdLast.Enabled = True
cmdFind.Enabled = True
cmdUpdate.Enabled = True
AddNew.Enabled = True

DoUpdate = False
DoAdd = False

txtEmpno.Enabled = True

End Sub
```

Deleting Records

Users can delete records by navigating to a particular record and clicking the Delete button. The application prompts the user to verify the deletion, then the application deletes the record using the `Delete` method. The program then refreshes the screen with the next record or with the previous record if the user deleted the last record in the dynaset.

The following example shows the delete-click event procedure:

```
Private Sub cmdDelete_Click()
'prompt user
Response = MsgBox("Do you really want to Delete?", vbYesNo + vbExclamation)

If Response = vbYes Then
    EmpDynaset.Delete
    'attempt to move to next record
    EmpDynaset.MoveNext
    If EmpDynaset.EOF Then 'If deleted last record
        EmpDynaset.MovePrevious
    End If
    Call EmpRefresh
End If
End Sub
```

Querying the Database

The employee application can be configured to allow users to search for particular records in the database. For demonstration purposes, a Find button is included to allow users to query only employee names. At any time, the user can enter the query in the Employee Name field, and click the Find button. The application then displays the result or displays a message if the name cannot be found.

To search for records, the `FindFirst` method is used. When the find operation succeeds, the record is displayed. If the find fails, a message is displayed. The current row is reset to the first row, because failures cause the dynaset to be BOF (beginning-of-file), effectively making the current row invalid.

The `Find_Click()` event procedure must do the following:

1. Build a find clause to find the record where the `ENAME` column matches the entered string.
2. Execute the find using the `FindFirst` method.
3. Display the record if it is found; if the record was not found, display a message and reset the current row to the first row.

The following example shows a typical find click event procedure:

```
Private Sub cmdFind_Click()
Dim FindClause As String
Dim SingleQuote As String

ErrMsg = ""
SingleQuote = "'"

On Error GoTo err_find
'build the find clause:
'Can make our query case insensitive by converting the names to upper case
'FindClause = "UPPER(ename) = " & SingleQuote & UCase(txtEname) & SingleQuote
FindClause = "ename = " & SingleQuote & txtEname & SingleQuote

EmpDynaset.DbFindFirst FindClause

If EmpDynaset.NoMatch Then
    MsgBox "Could not find record matching Employee Name " & txtEname
    EmpDynaset.DbMoveFirst
End If

Call EmpRefresh

Exit Sub
```

Using Batch Insert

A typical command to load the Batch Insert form looks like this:

```
Private Sub BInsert_Click()
  Load BatchInsert
  BatchInsert.Show
End Sub
```

See Also: "Programming a Batch Form" on page 6-16

Programming a Batch Form

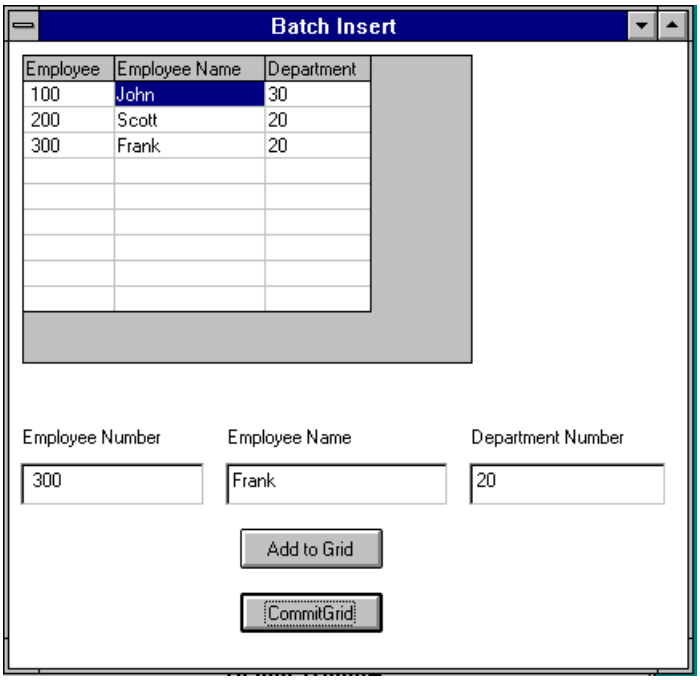
This section describes the Batch Insert Form and then describes the functions that it uses.

About the Batch Insert Form

The Batch Insert Form allows users to insert rows in a batch operation, that is, to insert more than one record into the database by using only one command. This feature is implemented using parameter arrays and SQL statements.

Table 6–4 shows a typical Batch Insert Form:

Figure 6–4 Batch Insert Form



Users navigate to the Batch Insert Form by clicking the Batch Insert button on the Employee Form. The Batch Insert Form has a grid that displays the entered data and a row of fields where the user enters each record. To keep the example simple, users are only allowed to enter information into the Employee Number, Employee Name, and Department Number fields.

Users enter records in the fields and click the Add to Grid button. The program displays the entered records in the grid. To insert the entire batch to the database, users click the CommitGrid button.

The Batch Insert Form uses three procedures. The `Form_Load()` procedure initializes the grid with the column headers. The `CmdAddtoGrid_click()` procedure copies the entered data from the fields to the grid. The `CommitGrid_Click()` procedure contains the parameter array and SQL statements used to make the batch insert.

These procedures are described as follows:

- Coding the Batch Insert Form_Load( ) Procedure
- Coding the CmdAddtoGrid( ) Procedure
- Coding the CommitGrid_Click( ) Procedure

Coding the Batch Insert Form_Load( ) Procedure

The following examples show how the Batch Insert `Form_Load()` procedure sets the column headings for the grid:

```
Private Sub Form_Load()
Grid1.Enabled = True
CurrRow = 0  'Top row
ReadRow = 0
ReadCol = 0

'Set column headings
```

```
Grid1.Row = CurrRow
Grid1.Col = 0
Grid1.Text = "Employee Number"

Grid1.Col = 1
Grid1.Text = "Employee Name"

Grid1.Col = 2
Grid1.Text = "Department Number"

NoOfCols = 3

CurrRow = CurrRow + 1

End Sub
```

Coding the CmdAddtoGrid( ) Procedure

The `CmdAddtoGrid_Click()` procedure copies the data entered in the fields to the next empty grid row. The global variable `CurrRow` always points to the first empty row in the grid.

The following example shows the `CmdAddtoGrid_Click()`:

```
Private Sub CmdAddtoGrid_Click()

'Update the grid
'Update Empno column
Grid1.Row = CurrRow
Grid1.Col = 0
Grid1.Text = txtEmpno
'Update Ename column
Grid1.Row = CurrRow
Grid1.Col = 1
Grid1.Text = txtEname
'Update Deptno column
Grid1.Row = CurrRow
Grid1.Col = 2
Grid1.Text = txtDeptno

'Increment CurrCol
CurrRow = CurrRow + 1

NoOfRows = CurrRow - 1

End Sub
```

Coding the CommitGrid_Click( ) Procedure

The `CommitGrid_Click()` procedure inserts the grid data into the database. To do so, this procedure creates a parameter array object for each column in the `EMP` table that corresponds to a column in the grid. The `OraParameters.AddTable()` method defines each parameter array. For example, a parameter array called `EMPNO_ARR` holds all Employee Number column elements.

After the parameter arrays are defined, the `Put_Value` method populates them with grid column elements.

To commit the parameter array elements to the database, this procedure uses the CreateSQL() method with a SQL INSERT statement containing the parameter arrays. Because the CreateSQL() method executes the SQL INSERT statement in addition to creating a SQL statement object, all column elements (parameter array elements) are inserted into the EMP table with this one statement.

If an error occurs during a SQL INSERT statement that contains parameter arrays, the SQL statement object is still created with no explicitly raised error. To identify such errors, always check the OraDatabase.LastServerErr and OraDatabase.LastServerErrText properties immediately after executing the CreateSQL method.

The CreateSQL method updates the database directly and has no effect on the dynaset. The EmpDynaset. Refresh method must used to refresh this dynaset so that it reflects the newly inserted records.

The CommitGrid_Click() event procedure must do the following:

1. Define a parameter array for each grid (database) column, using the AddTable method.
2. Copy grid column elements into parameter arrays, using the Put_Value method within a nested loop.
3. Create a SQL statement object using the CreateSQL method to insert parameter array elements into the EMP table.
4. Check the LastServerErrText and LastServerErr properties to catch SQL statement execution errors.
5. Refresh the global dyanset to reflect newly inserted records, using the Refresh method.

The following example shows a typical cmdCommitGrid_Click() procedure:

```
Private Sub cmdCommitGrid_Click()
Dim OraSqlStmt As Object
Dim OraPArray(2) As Object

On Error GoTo err_CommitGrid
ErrMsg = ""

'Define parameter arrays, one for each column
OraDatabase.Parameters.AddTable "EMPNO_ARR", ORAPARM_INPUT, ORATYPE_NUMBER, _
            NoOfRows
OraDatabase.Parameters.AddTable "ENAME_ARR", ORAPARM_INPUT, ORATYPE_VARCHAR2, _
            NoOfRows, 10
OraDatabase.Parameters.AddTable "DEPTNO_ARR", ORAPARM_INPUT, ORATYPE_NUMBER, _
            NoOfRows
If OraDatabase.LastServerErr <> 0 Or OraDatabase.LastServerErrText <> "" Then
Error 1
End If

'Initialize local array to hold parameter arrays
Set OraPArray(0) = OraDatabase.Parameters("EMPNO_ARR")
Set OraPArray(1) = OraDatabase.Parameters("ENAME_ARR")
Set OraPArray(2) = OraDatabase.Parameters("DEPTNO_ARR")

'Init the param array variables. Add loop to read thru grid ROWS
For ReadRow = 0 To (NoOfRows - 1)
Grid1.Row = ReadRow + 1
'Loop to read thru grid CELLS
    For ReadCol = 0 To NoOfCols - 1
```

```
        Grid1.Col = ReadCol
        OraPArray(ReadCol).Put_Value Grid1.Text, ReadRow
    Next ReadCol
Next ReadRow

'create a sqlstmt to insert array values into table
Set OraSqlStmt = OraDatabase.CreateSql("insert into emp(empno,ename,deptno)" & _
        "values(:EMPNO_ARR,:ENAME_ARR,:DEPTNO_ARR)", 0&)
If OraDatabase.LastServerErr <> 0 Or OraDatabase.LastServerErrText <> "" Then
  ErrMsg = OraDatabase.LastServerErrText
  Error 1
End If

'Refresh the Dynaset
EmpDynaset.Refresh

OraDatabase.Parameters.Remove "EMPNO_ARR"
OraDatabase.Parameters.Remove "ENAME_ARR"
OraDatabase.Parameters.Remove "DEPTNO_ARR"

Exit Sub

err_CommitGrid:
    If ErrMsg <> "" Then
        MsgBox ErrMsg
    Else
        MsgBox Error$
    End If

End Sub
```

7

Code Wizard for Stored Procedures

The Oracle Objects for OLE (OO4O) Code Wizard generates OO4O code that executes Oracle PL/SQL and Java stored procedures.

The wizard generates code into individual Microsoft Visual Basic or Active Server Page and VBScript subroutines from existing Oracle stored procedures and packages. Additionally, the wizard can generate complete implementations of COM Automation objects in the form of VB class files. The generated COM Automation object methods act as client stubs for the execution of stored procedures contained in a given package. All the OO4O code necessary for input/output parameter binding and stored procedure execution is automatically generated.

The wizard can be used as a command-line utility or as a Visual Basic add-in. The wizard automates the entire process of accessing stored procedures using COM interfaces, thereby significantly reducing development time and the likelihood of programming errors.

> **Note:** The Code Wizard requires Visual Basic 6.

This chapter contains these topics:

Oracle Objects for OLE Code Wizard Components

The OO4O Code Wizard includes the following components:

- A command line utility, `OO4OCodeWiz.exe`, that converts PL/SQL and Java stored procedures to OO4O code.
- A Visual Basic Add-in wizard that guides you through a series of steps to generate OO4O code for PL/SQL and Java stored procedures. The wizard displays Oracle packages and stored procedures from a tree control so that the user can choose which items to generate code.

Both of these components allow users to convert entire stored procedure packages to OO4O code.

Data Types Supported by the OO4O Code Wizard

The code wizard supports all data types, except for PL/SQL tables. When a PL/SQL table is used, an `unsupportedType` key word is used instead, and the generated code does not compile.

The output code may have to be modified for handling `Null` values. For example, when a VB variable is initialized to a parameter value, an `isNull()` check may have to be added if `Null` values are expected. `Null` values are correctly handled for VB variables of type `Variant` and `Object`.

Using the OO4O Code Wizard

The OO4O Code Wizard can be used as a command line utility or as a Visual Basic Add-in.

OO4O Code Wizard Command-Line Utility

The `OO4OCodeWiz.exe` is a command-line utility that generates a Visual Basic class, a Visual Basic file, or an Active Server Page/VB Script file from existing PL/SQL or Java stored procedures, as well as packages, within an Oracle database. Call the utility in the following manner:

```
OO4OCodeWiz [-o output_file] username/password@connect_string package
```

Where	Specifies the following
username	User name to log in to the database
password	Password for the user name
connect_string	Database connection string
package	Package name
stored_procedure	Stored procedure name (optional)

Example

```
OO4OCodeWiz -o empfile.asp scott/tiger@Exampledb employee.example
```

Option

Option	Description
-o	Specifies the output file name (optional)

Files Generated

The code wizard uses the information specified on the command line to determine which type of output file to generate.

If a file name and one of the permitted file extensions are specified, then they are used. In the preceding example, an ASP file is generated in the `empfile.asp` output. The user can specify the following extensions:

Extension	File Type Generated
.cls	VB class file
.bas	VB file
.asp	ASP or VB script file
.vbs	ASP or VB script file

If no file extension is specified, the following rules indicate what type of file is generated, depending on other command-line specifications.

- Package names without a stored procedure name generate a .cls file.
- Package names with procedure names generate a .bas file.

Table 7–1 and Table 7–2 provide examples.

Table 7–1 Package Name Without Stored Procedure Name

File Specified	Command	File Type Generated
File name with no file extension generates *filename.cls*.	OO4OCodeWiz -o empfile *scott/tiger@Exampledb* employee	empfile.cls
No file name or extension: generates *packagename.cls*.	OO4OCodeWiz scott/tiger@Exampledb *employee*	*employee*.cls
File name with file extension generates *filename.fileexten*.	OO4OCodeWiz -o empfile.asp scott/tiger@Exampledb *employee*	empfile.asp

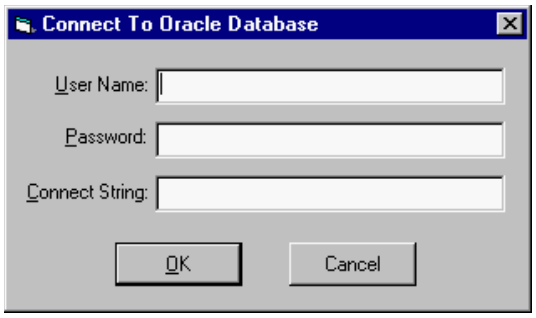
Table 7–2 Package Name With Stored Procedure Name

File Specified	Command	File Type Generated
File name with no file extension generates *filename.bas*.	OO4OCodeWiz -o empfile scott/tiger@Exampledb *employee*.example	empfile.bas
No file name or extension: generates *packagename.bas*.	OO4OCodeWiz scott/tiger@Exampledb *employee*.example	*employee*.bas
File name with file extension generates *filename.fileexten*.	OO4OCodeWiz -o empfile.asp scott/tiger@Exampledb *employee*.example	empfile.asp

OO4O Code Wizard Visual Basic Wizard Add-in

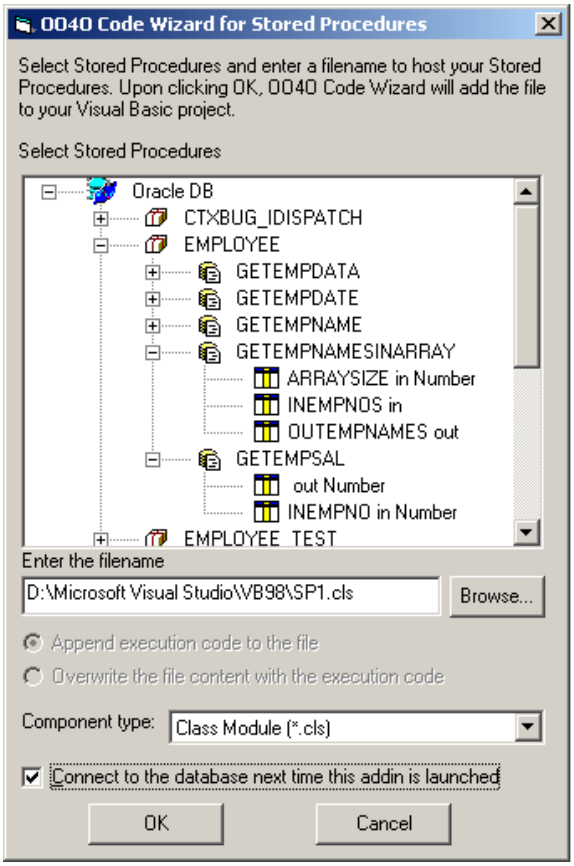
1. Launch the OO4O Code Wizard by selecting Oracle Code Wizard for Stored Procedures in the Add-Ins menu of Microsoft Visual Basic.

   The **Connect To Oracle Database** dialog box appears:



2. Enter the user name and password to connect to the database. A connection string is required if the database is not installed on the user's local computer.
3. Click **OK**.

   The wizard displays the Oracle packages and stored procedures available to the user in a tree.



4. Select one of the stored procedures or packages displayed.
5. Enter an output file name or click the **Browse...** button to navigate to a different directory in which to place the file.
6. Choose the file type from the **Component type** list. There are three choices: a VB class module (`*.cls`), a VB file (`*.bas`), or other. The **other** option generates a VB file (`*.bas`), but enables you to specify your own file extension.
7. Click **OK**.

A dialog box appears indicating that a new OO4O file was created.

8. Click **Yes** to create another file, or click **No** to return to Visual Basic.

Code Wizard Examples

The *ORACLE_BASE**ORACLE_HOME*\oo4o\codewiz\samples directory contains sample applications incorporating code generated by the wizard. The following examples show the generated VB code output from Oracle stored procedures using the OO4O code wizard:

- Accessing a PL/SQL Stored Function with Visual Basic and Active Server Pages
- Accessing a PL/SQL Stored Procedure Using the LOB Type with Visual Basic
- Accessing a PL/SQL Stored Procedure Using the VARRAY Type with Visual Basic
- Accessing a PL/SQL Stored Procedure Using the Oracle OBJECT Type with Visual Basic

Accessing a PL/SQL Stored Function with Visual Basic and Active Server Pages

This example shows a PL/SQL stored function, `GetEmpSal`, and then the Visual Basic (`*.cls`) file that the code wizard generates for it.

```
FUNCTION GetEmpSal (inEmpno IN NUMBER)
RETURN NUMBER is
  outEmpsal NUMBER(7,2);
BEGIN
  SELECT SAL into outEmpsal from EMP WHERE EMPNO = inEmpno;
  RETURN (outEmpsal);
END;
```

The generated code for the `GetEmpSal` stored function is:

```
Public Function GETEMPSAL(INEMPNO As Variant) As Variant
OraDatabase.Parameters.Add "INEMPNO", INEMPNO, ORAPARM_INPUT, 2
OraDatabase.Parameters.Add "result", 0, ORAPARM_OUTPUT
OraDatabase.Parameters("result").serverType = 2
OraDatabase.ExecuteSQL ("declare result Number; Begin :result := " & _
              "Employee.GETEMPSAL(:INEMPNO); end;")

OraDatabase.Parameters.Remove "INEMPNO"
GETEMPSAL = OraDatabase.Parameters("result").Value
OraDatabase.Parameters.Remove "result"
End Function
```

In a VB class, `OraDatabase` appears as an attribute of the class. This attribute has to be set before any methods of the class can be invoked. For a VB file (`*.bas`), the generated code for the `GetEmpSal` stored function is the same as the VB class file, except for the function declaration:

```
Public Function GETEMPSAL(INEMPNO As Variant, ByRef OraDatabase As OraDatabase)
...

End Function
```

For an ASP file (`*.asp`), the function declaration also differs for the `GetEmpSal` stored function as follows, but the body of the code remains the same:

```
Public Function GETEMPSAL(INEMPNO, ByRef OraDatabase)
...
```

```
End Function
```

Accessing a PL/SQL Stored Procedure Using the LOB Type with Visual Basic

The following example shows how a Visual Basic file accesses a PL/SQL stored procedure with LOBs:

```
PROCEDURE getchapter(chapno in NUMBER, chap out CLOB) is
BEGIN
SELECT chapters into chap from mybook where chapterno = chapno
   for update;
END;
```

The following shows the generated Visual Basic code for the `GETCHAPTER` stored procedure:

```
Public Sub GETCHAPTER(CHAPNO As Variant, ByRef CHAP As OraCLOB)
OraDatabase.Parameters.Add "CHAPNO", CHAPNO, ORAPARM_INPUT, 2
OraDatabase.Parameters.Add "CHAP", Null, ORAPARM_OUTPUT, 112
OraDatabase.ExecuteSQL ("Begin MYBOOKPKG.GETCHAPTER(:CHAPNO,:CHAP); end;")
Set CHAP = OraDatabase.Parameters("CHAP").Value
OraDatabase.Parameters.Remove "CHAPNO"
OraDatabase.Parameters.Remove "CHAP"
End Sub
```

Accessing a PL/SQL Stored Procedure Using the VARRAY Type with Visual Basic

The following example shows how a PL/SQL stored procedure uses the Oracle collection type `VARRAY`:

```
PROCEDURE getnames(deptid in NUMBER, name out ENAMELIST) is
BEGIN
    SELECT ENAMES into name from department where dept_id = deptid for update;
END;
```

The wizard generates the following Visual Basic code for this stored procedure:

```
Public Sub GETNAMES(DEPTID As Variant, ByRef NAME As OraCollection)
OraDatabase.Parameters.Add "DEPTID", DEPTID, ORAPARM_INPUT, 2
OraDatabase.Parameters.Add "NAME", Null, ORAPARM_OUTPUT, 247, "ENAMELIST"
OraDatabase.ExecuteSQL ("Begin DEPTPKG.GETNAMES(:DEPTID, :NAME); end;")
Set NAME = OraDatabase.Parameters("NAME").Value
OraDatabase.Parameters.Remove "DEPTID"
OraDatabase.Parameters.Remove "NAME"
End Sub
```

Accessing a PL/SQL Stored Procedure Using the Oracle OBJECT Type with Visual Basic

The following example shows how a PL/SQL stored procedure uses the Oracle object type:

```
PROCEDURE getaddress(person_name in varchar2, person_address out address) is
BEGIN
     SELECT addr into person_address from person_table where name =
           person_name for update;
END;
```

The wizard generates the following Visual Basic code for this stored procedure:

```
Public Sub GETADDRESS(PERSON_NAME As String, ByRef PERSON_ADDRESS As OraObject)
OraDatabase.Parameters.Add "PERSON_NAME", PERSON_NAME, ORAPARM_INPUT, 1
OraDatabase.Parameters.Add "PERSON_ADDRESS", Null, ORAPARM_OUTPUT, _
            108, "ADDRESS"
OraDatabase.ExecuteSQL ("Begin PERSONPKG.GETADDRESS(:PERSON_NAME," & _
            ":PERSON_ADDRESS); end;")
Set PERSON_ADDRESS = OraDatabase.Parameters("PERSON_ADDRESS").Value
OraDatabase.Parameters.Remove "PERSON_NAME"
OraDatabase.Parameters.Remove "PERSON_ADDRESS"
End Sub
```

8

Introduction to Automation Objects

This chapter introduces commonly used OO4O Automation Objects.

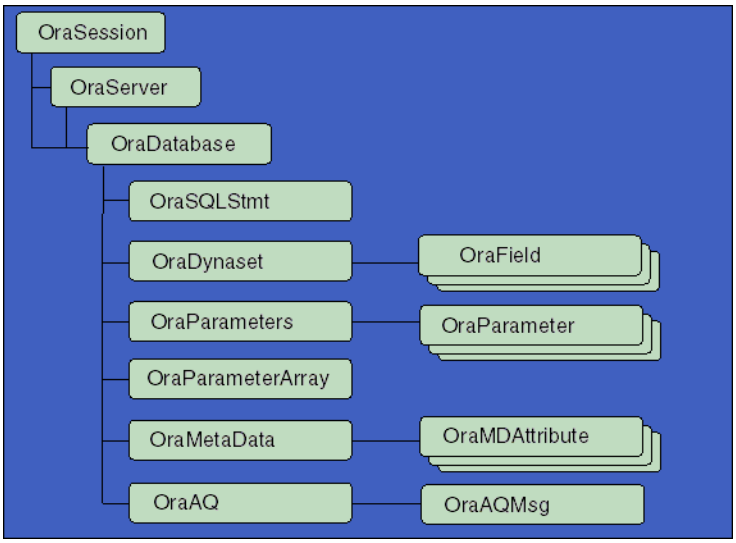
This chapter contains these topics:

Overview of Automation Objects

The OO4O operational hierarchy of the objects expresses has-a and belongs-to relationships.

Figure 8–1 shows the operational hierarchy.

Figure 8–1 OO4O Automation Objects



The Automation objects diagram illustrates this hierarchy.

OraSession Object Overview

The `OraSession` object is returned when an instance of the OO4O Automation Server is created. It mainly serves as an interface for establishing connections to Oracle databases. It also contains methods for starting, committing, and canceling transactions on the connections contained in the `OraDatabase` objects created. The following Visual Basic example creates an instance of the OO4O Automation Server.

```
'OracleInProcServer.XOraSession is the symbolic name for a
'globally unique component identifier.
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")
```

See Also:

- OraSession Object on page 9-58
- "Accessing the Oracle Objects for OLE Automation Server" on page 3-1
- "Connecting to Oracle Database" on page 3-2

OraServer Object Overview

The `OraServer` object represents a physical connection to an Oracle database instance. It provides a method, `OpenDatabase`, for creating user sessions, which represents `OraDatabase` objects. It makes it possible to do "connection multiplexing."

See Also:

- OraServer Object on page 9-56
- "Accessing the Oracle Objects for OLE Automation Server" on page 3-1
- "Using OraServer for Connection Multiplexing" on page 3-3

OraDatabase Object Overview

The OraDatabase object represents a user connection to an Oracle database instance, and provides methods to execute SQL statements and PL/SQL code. The OraDatabase object is returned by the OpenDatabase method of the OraSession or the OraServer object.

The following example illustrates the use of the OpenDatabase method of the OraSession. OraDatabase objects created by this method contain a distinct physical connection to an Oracle database.

```
'Establish a connection to the ExampleDb database
Set hrDBSession = OO4OSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)
```

The following example demonstrates how a physical network connection to an Oracle database can be shared by multiple user sessions. Using a single connection that is shared by multiple user sessions results in reduced resource usage in an Oracle Database and can increase scalability.

```
'Create a server connection

Set hrDBServer = CreateObject("OracleInProcServer.XOraServer")
Set hrDBServer = oo4o.Open("ExampleDb")
Set userSession1 = hrDBServer.OpenDatabase("scott/tiger", 0)

'execute queries ...
Set userSession2= hrDBServer.OpenDatabase("scott/tiger", 0)

'execute queries ...
```

See Also: OraDatabase Object on page 9-27

OraDynaset Object Overview

An OraDynaset object represents the result set of a SQL SELECT query or a PL/SQL cursor variable returned from a stored procedure or function. It is essentially a client-side scrollable and updatable cursor that allows for browsing the set of rows generated by the query it executes. It is created by the CreateDynaset or CreateCustomDynaset method of an OraDatabase interface.

The following Visual Basic example executes a query, loops through the result set, and displays values of columns returned.

```
Set employees = OraDatabase.CreateDynaset("select empno, ename from emp", 0&)

'While there are more rows
while not employees.EOF

'Display the values of empno and ename column of the current row
msgbox employees("empno") & employees("ename")

'Move to the next row
employees.MoveNext
wend
```

See Also:

- OraDynaset Object on page 9-30
- CreateCustomDynaset Method on page 10-80
- CreateDynaset Method on page 10-85

OraField Object Overview

The `OraField` object is an abstraction of a column in an `OraDynaset` object. It contains the value as well as the metadata that describes a column of the current row in the dynaset. In the previous example for the `OraDynaset` object, the `Field` interface for `empno` can be obtained using this additional code:

```
set empno = employees.Fields("empno")
msgbox "Employee Number: " & empno.Value
```

`OraFields` is a collection object representing all columns in the current row.

`OraField` objects can represent instances of any data type supported by Oracle Database. This includes all primitive types, such as `VARCHAR2`, `NUMBER`, `INT`, and `FLOAT`, as well all the object-relational types introduced in Oracle8*i*.

See Also:

- OraField Object on page 9-33
- "Support for Oracle Object-Relational and LOB Data Types" on page 4-1

OraParameters Object Overview

The `OraParameters` object is a collection container for `OraParameter` objects. An `OraParameter` object is used to supply data values for placeholders used in the SQL statements or PL/SQL blocks at run time. It can be used to provide input values as well as contain values that are returned from the database. The following sample creates two parameter objects and uses them in an update query.

```
OraDatabase.Parameters.Add "SALARY", 4000, ORAPARM_INPUT
OraDatabase.Parameters.Add "ENAME", "JONES", ORAPARM_INPUT
Set updateStmt = OraDatabase.CreateSQL("update emp set sal = :SALARY" & _
             "where ename = :ENAME ")
```

See Also: OraParameters Collection on page 9-68

OraParameter Object Overview

`OraParameter` objects can contain values for all the data types supported by Oracle9*i* including object-relational data types. They can be passed as input or output arguments to PL/SQL stored procedures and functions. The values of the `OraParameter` objects can also represent PL/SQL cursors in the form of `OraDynaset` objects.

See Also:

- OraParameter Object on page 9-50
- "PL/SQL Support" on page 3-9

OraParamArray Object Overview

An `OraParamArray` object provides the mechanism for binding and fetching an array of values. It is typically used for performing bulk inserts and updates.

```
'Create a table
OraDatabase.ExecuteSQL ("create table part_nos(partno number," & _
             "description char(50), primary key(partno))")

'Create two parameter arrays of size 10 to hold values for
'part numbers (size 22 bytes), and their description (50 bytes long).
OraDatabase.Parameters.AddTable "PARTNO", ORAPARM_INPUT, ORATYPE_NUMBER, 10, 22
OraDatabase.Parameters.AddTable "DESCRIPTION", ORAPARM_INPUT, _
            ORATYPE_CHAR, 10, 50

'Initialize the arrays
For I = 0 To 10
   OraDatabase.Parameters("PARTNO").put_Value = I, I
   OraDatabase.Parameters("DESCRIPTION ") = "some description", I
Next I

'Execute the query
Set OraSqlStmt = OraDatabase.CreateSql("insert into " & _
            "part_nos(partno, description) values(:PARTNO,:DESCRIPTION)", 0&)
```

See Also: OraParamArray Object on page 9-47

OraSQLStmt Object Overview

The `OraSQLStmt` object is typically used for executing non-select SQL queries and PL/SQL blocks. The following line of code executes an update query and displays the number of rows affected.

```
Set updateStmt = OraDatabase.CreateSQL("update emp set sal = 3000" & _
                "where ename = 'JONES' ")
MsgBox updateStmt.RecordCount
```

The `OraSQLStmt` object (`updateStmt`) can be used later to execute the same query with a different value for the `:SALARY` placeholder. For example:

```
OraDatabase.Parameters("SALARY").value = 200000
updateStmt.Parameters("ENAME").value = "KING"
updateStmt.Refresh
```

See Also: OraSQLStmt Object on page 9-60

9

Server Objects

This chapter describes the Oracle Objects for OLE Server Objects.

See Also:

- "Overview of Oracle Objects for OLE" on page 1-1
- "Oracle Objects for OLE In-Process Automation Server" on page 1-2
- "Using Automation Clients Overview" on page 2-1
- "Required Setups" on page 1-5

This chapter contains these topics:

- OraRef Object
- OraServer Object
- OraSession Object
- OraSQLStmt Object
- OraSubscription Object
- OraTimeStamp Object
- OraTimeStampTZ Object
- OraConnections Collection
- OraFields Collection
- OraParameters Collection
- OraSessions Collection
- OraSubscriptions Collection

OraAQ Object

Description

An `OraAQ` object is instantiated by invoking the `CreateAQ` method of the `OraDatabase` interface. It represents a queue that is present in the database.

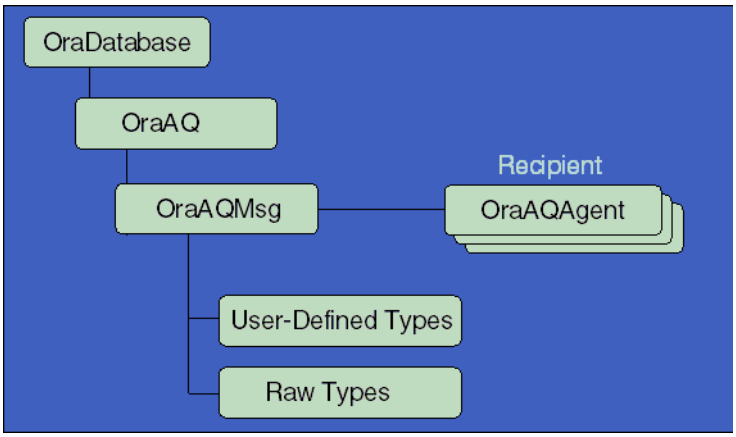
Remarks

Oracle Objects for OLE provides interfaces for accessing Oracle Database Advanced Queuing (AQ) feature. It makes AQ accessible from popular COM-based development environments such as Visual Basic.

The `OraAQ` Automation interface provides methods for enqueuing and dequeuing messages (encapsulated in the `OraAQMsg` object). It also provides a method for monitoring queues for message arrivals.

Client applications provide a `Dispatch` interface to the monitor. The monitor checks the queue for messages that meet the application criteria. It then invokes the `NotifyMe` method of the `Dispatch` interface when these messages are dequeued.

The following diagram illustrates the OO4O AQ Automation objects and their properties.



Properties

- Consumer (OraAQ) Property on page 11-28
- Correlate (OraAQ) Property on page 11-29
- DequeueMode (OraAQ) Property on page 11-47
- DequeueMsgId (OraAQ) Property on page 11-48
- Navigation (OraAQ) Property on page 11-109
- RelMsgId (OraAQ) Property on page 11-131
- Visible (OraAQ) Property on page 11-186
- Wait (OraAQ) Property on page 11-187

Methods

- AQMsg (OraAQ) Method on page 10-33

- Enqueue (OraAQ) Method on page 10-141
- Dequeue (OraAQ) Method on page 10-122
- MonitorStart (OraAQ) Method on page 10-196
- MonitorStop (OraAQ) Method on page 10-198

Examples

Example: Enqueuing Messages

Enqueuing messages of type RAW

"Enqueuing Messages of Type RAW" on page 10-141

Enqueuing messages of Oracle object types

"Enqueuing Messages of Oracle Object Types" on page 10-142

Example: Dequeuing messages

NOTE: The following code samples serve as models for dequeuing messages.

A complete AQ sample can be found in `\OO4O\VB\SAMPLES\AQ`

Dequeuing messages of the RAW type

"Example: Dequeuing Messages of RAW Type" on page 10-122

Dequeuing messages of Oracle object types

"Example: Dequeuing Messages of Oracle Object Types" on page 10-123

Example: Monitoring messages

See "Monitoring Messages" on page 4-21 for examples illustrating the use of the `MonitorStart` and `MonitorStop` methods.

See Also:

- OraAQAgent Object on page 9-5
- OraAQMsg Object on page 9-6
- *Oracle Streams Advanced Queuing User's Guide* for a detailed description of Oracle Advanced Queuing

OraAQAgent Object

Description

The `OraAQAgent` object represents a message recipient and is only valid for queues that allow multiple consumers.

Remarks

An `OraAQAgent` object can be instantiated by invoking the `AQAgent` method. For example:

```
Set agent = qMsg.AQAgent(name)
```

Methods

None.

Properties

- Address (OraAQAgent) Property on page 11-7
- Name (AQAgent) Property on page 11-103

Example

The following Visual Basic example illustrates a simple use of the advanced queuing feature. A message of a user-defined type, `MESSAGE_TYPE`, is enqueued into a queue, `msg_queue`, that supports multiple consumers.

```
Dim q as OraAQ
Dim qMsg as OraAQMsg
Dim agent as OraAQAgent
Set q = OraDatabase.CreateAQ("msg_queue")
Set qMsg = q.AQMsg(1,"MESSAGE_TYPE")

'To add SCOTT as a recipient for the message,
Set agent = qMsg.AQAgent("SCOTT")

'To enqueue,
q.Enqueue
```

See Also:

- *Oracle Streams Advanced Queuing User's Guide* for a detailed description of Oracle Advanced Queuing
- OraAQ Object on page 9-3
- OraAQMsg Object on page 9-6

OraAQMsg Object

Description

The `OraAQMsg` object encapsulates the message to be enqueued or dequeued. The message can be of any user-defined or raw type.

Properties

- Correlation (OraAQMsg) Property on page 11-30
- Delay (OraAQMsg) Property on page 11-46
- ExceptionQueue Property on page 11-58
- Expiration (OraAQMsg) Property on page 11-60
- Priority (OraAQMsg) Property on page 11-126
- Value (OraAQMsg) Property on page 11-176

Methods

- AQAgent (OraAQMsg) Method on page 10-32

See Also:

- *Oracle Streams Advanced Queuing User's Guide* for a detailed description of Oracle Advanced Queuing
- OraAQ Object on page 9-3
- OraAQAgent Object on page 9-5

OraAttribute Object

Description

The OraAttribute object represents an attribute of a Value or REF instance of an OraObject or an OraRef.

Remarks

The OraAttribute object can be accessed from the OraObject or OraRef object by creating a subscript that uses ordinal integers or by using the name attribute.

See the Value (OraAttribute) property for a table that identifies the attribute type and the return value of the Value property of the OraAttribute object:

Properties

- Value (OraAttribute) Property on page 11-175
- Name (OraAttribute) Property on page 11-104
- Type (OraAttribute) Property on page 11-166

Methods

None.

Examples

The following example accesses the attributes of the ADDRESS value instance in the server. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim Address As OraObject
Dim City As OraAttribute
Dim State As OraAttribute

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

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

'create a dynaset object from person_tab
Set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab", 0&)

'retrieve an address column from person_tab
'the Value property of OraField object returns Address OraObject
Set Address = OraDynaset.Fields("Addr").Value

'access the City attribute object
Set City = Address("City")

' display the value of City attribute Object
MsgBox City.Value
```

```
'access the State attribute object
Set State = Address("State")

'display the value of State attribute Object
MsgBox State.Value
```

See Also:

- OraCollection Object on page 9-19
- OraParameter Object on page 9-50
- OraObject Object on page 9-43
- OraRef Object on page 9-52

OraBFILE Object

Description

The `OraBFile` interface in OO4O provides methods for performing operations on the `BFILE` LOB data type in the database.

Remarks

The `BFILE` types are large binary data objects stored in operating system files (external) outside of the database tablespaces.

Properties

- DirectoryName Property on page 11-49
- FileName Property on page 11-68
- Exists Property on page 11-59
- IsNull (OraLOB/BFILE) Property on page 11-80
- IsOpen (OraBFILE) Property on page 11-83
- Offset (OraLOB/BFILE) Property on page 11-112
- PollingAmount Property on page 11-125
- Size (OraLOB and OraBFILE) Property on page 11-145
- Status (OraLOB/BFILE) Property on page 11-154

Methods

- Clone (OraLOB/BFILE) Method on page 10-53
- Close (OraBFILE) Method on page 10-64
- CloseAll (OraBFILE) Method on page 10-65
- Compare (OraLOB) Method on page 10-68
- CopyToFile (OraLOB/BFILE) Method on page 10-76
- MatchPos (OraLOB/BFILE) Method on page 10-192
- Open (OraBFILE) Method on page 10-211
- Read (OraLOB/BFILE) Method on page 10-221

Examples

See "Schema Objects Used in LOB Data Type Examples" on page A-3 for schema objects that are used in the OraLOB/`BFILE` examples.

NOTE: To add the required tables for the following examples, run the `lob.sql` file in the `\OO4O\VB\SAMPLES\LOB` directory.

Example: Accessing the BFILE Value

`BFILE` data can be read using the `Read` method. The `OraBFILE` object allows piecewise read operations. Before reading the `BFILE` content, the `BFILE` file should be opened using the `Open` method.

```
Dim PartColl as OraBFile
```

```
Dim buffer As Variant

'Create a Dynaset containing a BLOB and a CLOB column
set part = OraDatabase.CreateDynaset ("select * from part",0)
Set PartColl = part.Fields("part_collateral").Value

'open the bfile for read operation
PartColl.Open

'read the entire bfile
amount_read = PartColl.Read(buffer)

'close the bfile
PartColl.Close
```

Example: Reading and Inserting BFILEs Using Dynasets

To modify the directory and file names of the `BFILE` value of an `OraBFILE` object, first obtain a lock and then use the `DirectoryName` and `FileName` properties.

To insert a new row containing a `BFILE` column, initialize the `BFILE` column with new directory and file name values using the `DirectoryName` and `FileName` properties.

```
Dim PartColl as OraBFile
Dim buffer As Variant

'Create a Dynaset containing a BLOB and a CLOB column
set part = OraDatabase.CreateDynaset ("select * from part",0)
Set PartColl = part.Fields("part_collateral").Value

'insert a new BFILE in the part_collateral column
part.AddNew

'Directory objects will be upper-case by default
    PartColl.DirectoryName = "NEWDIRECTORYNAME"
    PartColl.FileName = "NewPartCollatoral"
part.Update

'move to the newly added row
part.MoveLast

'open the Bfile for read operation
PartColl.Open

'read the entire bfile
amount_read = PartColl.Read(buffer)

'close the Bfile
PartColl.Close
```

See Also:

- OraBLOB, OraCLOB Objects on page 9-11
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for a detailed description of Oracle `BFILE` types

OraBLOB, OraCLOB Objects

Description

The OraBLOB and OraCLOB interfaces in OO4O provide methods for performing operations in a database on the large object data types BLOB, CLOB, and NCLOB. In this developer's guide, BLOB, CLOB, and NCLOB data types are also referred to as LOB data types.

OO4O supports the creation of temporary BLOB or CLOB types that can be manipulated and then bound to SQL statements or PL/SQL blocks, or copied into permanent LOBs.

Remarks

LOB data is accessed using the Read and CopyToFile methods.

LOB data is modified using the Write, Append, Erase, Trim, Copy, CopyFromFile, and CopyFromBFile methods. A row lock must be obtained before modifying the contents of a LOB column in a row. If the LOB column is a field of an OraDynaset object, then the lock is obtained by invoking the Edit method.

None of the LOB operations are allowed on NULL LOBs. To avoid errors, use the IsNull property to detect NULL LOBs. To perform write operations on a LOB that is null, first the LOB column must be initialized with an Empty value.

To insert a new row having a LOB column, first initialize the LOB column with an Empty value by setting the Value property of the OraField or OraParameter object to the keyword Empty and commit the change to the database. The newly updated Empty LOB must be selected again from the database before it can be used. This is done automatically in the case of the OraDynaset object: If a LOB field in an OraDynaset object is set to Empty and the Update method is called, OO4O automatically reselects the Empty LOB into the dynaset making it available for use in subsequent write operations.

There are two modes of operation for read and write operations for LOBs.

1. Multiple-piece read/write operations

   In this mode, the total amount of data to be read or written is more than the size of the buffer for an individual read/write operation. Rather than make a complete round-trip for each operation, the pieces are streamed. To begin the multiple piece operation, the PollingAmount property is first set to the total amount of data to be read or written. The Offset property is set at this time to specify the initial offset for the first piece read/write operation. The offset is automatically incremented after the first read/write operation, and cannot be changed again until the multiple piece operation has completed. The Status property must be checked for the success of each piecewise operation and the operation must continue until all the pieces are read or written (it cannot be aborted). To start another multiple-piece read/write operation on the same LOB, the PollingAmount property has to be reset to the desired amount. See "Example: Multiple-Piece Read of a LOB" on page 10-221.

2. Single-piece read/write operation

   In this mode, the reading and writing of data occurs in one operation. This mode is enabled when the PollingAmount property is set to 0. See "Example: Single-Piece Read of a LOB" on page 10-222.

The Offset property in both modes of operation is 1-based.

By design, LOBs cannot span transactions started by SELECT .. FOR UPDATE, INSERT, and UPDATE statements. Selecting or modifying LOB values using these SQL statements makes LOBs invalid outside the current transaction. In Oracle Objects for OLE, transactions can be started and ended in the following ways.

1. Dynaset Edit/Update method

   The Edit method executes the SELECT FOR UPDATE statement to lock the row and start the transaction. The Update method ends the transaction. If the LOB column value is modifed between the Edit and Update pair, OO4O reselects the value of LOB column after the Update call. This is transparent to the user. Note that OO4O does not reselect the LOB value if the LOB is an attribute of an Oracle objects instance or element of an Oracle collection. If the transaction is started by the OraSession/OraDatabase or OraServer object and the LOB data is modified between the Edit and Update methods, OO4O does not reselect the LOB value from the database. LOBs are invalid after committing transactions initiated by OraSession/OraDatabase or OraServer objects.

   See "Example: Dynasets Containing LOBs and Transactions" on page 9-16.

2. Executing an INSERT or UPDATE statement through the ExecuteSQL or CreateSQL method.

   An INSERT or UPDATE statement starts the transaction, and the transaction is implicitly ended by Oracle Objects for OLE (auto-commit). If a statement has a LOB output bind parameter, as in the case of the RETURNING .. INTO clause, then it will become invalid after the ExecuteSQL or CreateSQL method is executed To avoid this, the user must execute these statement between the BeginTrans/CommitTrans pair of OraSession, OraServer or OraDatabase objects.

   See "Example: INSERT or UPDATE Statements with LOBs and Transactions" on page 9-16.

See Also:

- "Using Large Objects (LOBs)" on page 4-3 for more information about LOB operations and LOB performance issues
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for a detailed description of Oracle LOBs

Properties

- IsNull (OraLOB/BFILE) Property on page 11-80
- PollingAmount Property on page 11-125
- Offset (OraLOB/BFILE) Property on page 11-112
- Size (OraLOB and OraBFILE) Property on page 11-145
- Status (OraLOB/BFILE) Property on page 11-154

Methods

- Append (OraLOB) Method on page 10-27
- Clone (OraLOB/BFILE) Method on page 10-53
- Compare (OraLOB) Method on page 10-68

- Copy (OraLOB) Method on page 10-72
- CopyFromFile (OraLOB) Method on page 10-73
- CopyFromBFILE (OraLOB) Method on page 10-75
- CopyToFile (OraLOB/BFILE) Method on page 10-76
- DisableBuffering (OraLOB) Method on page 10-129
- EnableBuffering (OraLOB) Method on page 10-139
- Erase (OraLOB) Method on page 10-143
- FlushBuffer (OraLOB) Method on page 10-154
- MatchPos (OraLOB/BFILE) Method on page 10-192
- Read (OraLOB/BFILE) Method on page 10-221
- Trim (OraLOB) Method on page 10-254
- Write (OraLOB) Method on page 10-261

Examples

See "Schema Objects Used in LOB Data Type Examples" on page A-3 for schema objects that are used in the OraLOB and `BFILE` examples.

Example: Accessing a LOB Value

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim PartImage as OraBlob
Dim buffer As Variant

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

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

'execute the select statement
set OraDynaset = OraDatabase.CreateDynaset ("select * from part",0&)

'retrieve photo field from the dynaset
set PartImage = OraDynaset.Fields("part_image").Value

'read the entire LOB column in one piece into the buffer
amount_read = PartImage.Read(buffer, 10)
'use the buffer for internal processing
```

Example: Modifying a LOB Value

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim  PartDesc as OraClob
Dim buffer As String
```

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

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

'execute the select statement
set OraDynaset = OraDatabase.CreateDynaset ("select * from part",0&)
set PartDesc = OraDynaset.Fields("part_desc").Value

'To get a free file number
FNum = FreeFile
'Open the file for reading
Open "partdesc.dat" For Binary As #FNum

'Allocate buffer to the size of file FNum and read the entire file
buffer = String$(LOF(FNum), 32)
Get #FNum, , buffer

'lock the row for write operation
OraDynaset.Edit
amount_written = PartDesc.Write(buffer)

'commit the operation and release the lock
OraDynaset.Update
Close FNum
```

Example: Inserting LOBs Using Dynasets

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim Part As OraDynaset
Dim PartImage as OraBLOB
Dim ImageChunk() As Byte
Dim amount_written As Long

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

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

'Create a Dynaset containing a BLOB and a CLOB column

set part = OraDatabase.CreateDynaset ("select * from part",0)
set PartImage = part.Fields("part_image").Value

'First insert Empty LOB in the part_image column
part.AddNew
    part.Fields("part_id").Value = 1234
    part.Fields("part_image").Value = Empty
part.Update

'move to the newly added row
Part.MoveLast

'To get a free file number
```

```
FNum = FreeFile

'Open the file for reading PartImages
Open "part_picture.gif" For Binary As #FNum

'Re adjust the buffer size to hold entire file data

Redim ImageChunk(LOF(FNum))

'read the entire file and put it into buffer
Get #FNum, , ImageChunk

'call dynaset's Edit method to lock the row
part.Edit
amount_written = OraBlob.Write(ImageChunk)
part.Update

'close the file
Close FNum
```

Example: Inserting LOBs Using an OraParameter Object

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraBlob As OraBlob
Dim ImageChunk() As Byte
Dim amount_written As Long

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

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)
Set OraParameters = OraDatabase.Parameters
OraParameters.Add "PartImage", Empty,ORAPARM_OUTPUT
OraParameters("PartImage").ServerType = ORATYPE_BLOB

'BeginTrans needs to be called since LOB locators become
'invalid after the ExecuteSQL call
OraSession.BeginTrans
OraDatabase.ExecuteSQL ("insert into part values (1234,'Oracle  Application'," & _
                "EMPTY_BLOB(),NULL,NULL) RETURNING part_image INTO :PartImage")
set PartImage = OraDatabase.Parameters("PARTIMAGE").Value

FNum = FreeFile
'Open the file for reading PartImages
Open "part_picture.gif" For Binary As #FNum

'read the file and put it into buffer
Redim ImageChunk(LOF(FNum))
Get #FNum, , ImageChunk

Set OraBlob = OraDatabase.Parameters("PartImage").Value
amount_written = OraBlob.Write(ImageChunk, 10, ORALOB_ONE_PIECE)

' commit the transaction and close the file
OraSession.CommitTrans
Close FNum
```

Example: Dynasets Containing LOBs and Transactions

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraBlob As OraBlob
Dim PartImage as OraBLOB
Dim ImageChunk() As Byte
Dim amount_written As Long

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

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

'Create a Dynaset containing a BLOB and a CLOB column
set part = OraDatabase.CreateDynaset ("select * from part " & _
                              "where part_id = 1234",0)
set PartImage = part.Fields("part_image").Value

'To get a free file number
FNum = FreeFile

'Open the file for reading PartImages
Open "c:\part_picture.gif" For Binary As #FNum
Redim ImageChunk(LOF(FNum))

'read the file and put it into buffer
Get #FNum, , ImageChunk

'starts the transaction on OraSession
OraSession.BeginTrans

'call dynaset's Edit method to lock the row
part.Edit
Set OraBlob = PartImage
amount_written = OraBlob.Write(ImageChunk, 10, ORALOB_ONE_PIECE)
part.Update

'ends the transaction
OraSession.CommitTrans

'the following lines of code will raise error
'LOB locator cannot span transaction'
msgbox Partimage.Size
Close FNum
```

Example: INSERT or UPDATE Statements with LOBs and Transactions

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim ImageChunk() As Byte
Dim amount_written As Long

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

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

Set OraParameters = OraDatabase.Parameters
OraParameters.Add "PartImage", Empty,ORAPARM_OUTPUT
OraParameters("PartImage").ServerType = ORATYPE_BLOB

'Create a Dynaset containing a LOB,column
OraDatabase.ExecuteSQL ("insert into part values (1234,'Oracle Application'," & _
                  "EMPTY_BLOB(),NULL,NULL) RETURNING part_image INTO :PartImage")
set PartImage = OraDatabase.Parameters("PARTIMAGE").Value

'the following lines of code will raise error
'LOB locator cannot span transaction'
msgbox Partimage.Size
```

Example: Using the CopyToFile Method

See "Example:Using the CopyToFile Method" on page 10-76.

Example: Using the CopyFromFile Method

See "Example: Using the CopyFromFile Method" on page 10-73.

Example: Multiple-Piece Read of a LOB

See "Example: Multiple-Piece Read of a LOB" on page 10-221.

Example: Single-Piece Read of a LOB

See "Example: Single-Piece Read of a LOB" on page 10-222.

Example: Multiple-Piece Write of a LOB

See "Multiple-Piece Write of a LOB Example" on page 10-262.

Example: Single-Piece Write of a LOB

See "Single-Piece Write of a LOB Example" on page 10-263.

Example: Passing a Temporary CLOB to a Stored Procedure

See "Example: Passing a Temporary CLOB to a Stored Procedure" on page 10-114.

See Also:

- OraBFILE Object on page 9-9
- CreateTempBLOB/CLOB Method on page 10-114

OraClient Object

Description

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

Remarks

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

Properties

- Name Property on page 11-101
- Sessions Property on page 11-142

Methods

- CreateSession Method on page 10-109

See Also:

- OraSession Object on page 9-58
- OraSessions Collection on page 9-69

OraCollection Object

Description

The OraCollection interface represents Oracle collection types, such as variable-length arrays (VARRAYs) and nested tables.

Remarks

A collection is an ordered group of elements, all of the same type. For example, the students in a class or the grades for each student in a class. Each element has a unique subscript, called an index, that determines its position in the collection.

The collection type nested table is viewed as a table stored in the column of database tables. When retrieved, rows of a nested table are given consecutive subscripts that start at 1. Individual rows are accessed using an array-like access.

The collection type VARRAY is viewed as an array stored in the column of database tables. To reference an element in a VARRAY data type, standard subscripting syntax can be used. For example, Grade(3) references the third element in the VARRAY data type named Grades.

The OraCollection provides methods for accessing and manipulating an Oracle collection. Implicitly an OraCollection object contains an OLE Automation collection interface for accessing and manipulating (updating and inserting) individual elements of an Oracle collection. Individual elements can be accessed by using a subscript. An OraCollection element index starts at 1.

Element values are retrieved as Variant types. The Variant type of the element depends on the element type of the collection. Element values can be Null and can be set to Null. For elements of type objects and REFs, element values are returned as corresponding OO4O objects for that type. VARRAYs and nested tables do not support the elements of LOBs, VARRAYs, and Nested tables.

Table 9–1 lists the element type and return value of the elements.

Table 9–1 Element Type and Return Value of Elements

Element Type	Element Value
Object	OraObject
REF	OraRef
Date	String
Number	String
CHAR, VARCHAR2	String
Real	Real
Integer	Integer

Element values are converted into a Variant SAFEARRAY format using the SafeArray property. Only elements of primitive types are supported. A Variant SAFEARRAY index starts at 0.

The CreateOraObject method on the OraDatabase object returns the OraCollection object. The Oracle collection associated with this OraCollection object is created in the client-side object cache.

For information about creating a dynaset from a collection, see to "Creating a Dynaset from an OraCollection Object" on page 4-18.

Properties

- BOC Property on page 11-10
- ElementType Property on page 11-54
- EOC Property on page 11-55
- IsLocator (OraCollection) Property on page 11-77
- IsNull (OraCollection) Property on page 11-79
- MaxSize (OraCollection) Property
- SafeArray (OraCollection) Property on page 11-133
- Size (OraCollection) Property on page 11-144
- TableSize (OraCollection) Property on page 11-157
- Type (OraCollection) Property on page 11-167

Methods

- Append (OraCollection) Method on page 10-25
- Clone (OraCollection) Method on page 10-54
- CreateIterator Method on page 10-88
- Delete (OraCollection) Method on page 10-118
- DeleteIterator Method on page 10-121
- ElementValue Method on page 10-138
- Exist (OraCollection) Method on page 10-147
- InitIterator Method on page 10-171
- IterNext Method on page 10-187
- IterPrev Method on page 10-188
- Trim (OraCollection) Method on page 10-252

Examples

Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3 for schema objects that are used in the `OraCollection` examples.

Example: Accessing Collection Elements

The following example illustrates how to access collection elements.

OraDynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameList as OraCollection

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

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

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'retrieve a Enames column from Department.
'Here Value property of OraField object returns EnameList OraCollection
set EnameList = OraDynaset.Fields("Enames").Value

'access the first element of EnameList
msgbox  EnameList(1)

'move to next to row
OraDynaset.MoveNext

'access all the elements of EnameList for the second row
For index = 1 To EnameList.Size
        msgbox EnameList(index)
Next Index
```

OraParameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim EnameList as OraCollection

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

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

'create an  OraParameter object representing EnameList collection bind Variable
OraDatabase.Parameters.Add "ENAMES", Null, ORAPARM_OUTPUT, _
                ORATYPE_VARRAY,"ENAMELIST"

'execute the sql statement which selects ENAMES VARRAY  from the department  table

OraDatabase.ExecuteSQL ("BEGIN select enames  into :ENAMES  from department " & _
                  "where  dept_id = 10;  END;")

'get the EnameList collection  from OraParameter
set EnameList = OraDatabase.Parameters("ENAMES").Value

'access all the elements of EnameList
For index = 1 To EnameList.Size
        msgbox EnameList(index)
Next Index
```

Example: Modifying Collection Elements

The following example illustrates how to modify collection elements.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameList as OraCollection

'create the OraSession Object.
```

```
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

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

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'retrieve a Enames column from Department. Here Value property of OraField object
'returns EnameList OraCollection

set EnameList = OraDynaset.Fields("Enames").Value

'lock the row for editing and set the 2nd element of the EnameList to new value
OraDynaset.Edit
EnameList(2) = "Eric"
OraDynaset.Update
```

Example: Inserting in a Collection

The following example illustrates how to insert elements into an Oracle collection.

OraDynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameListNew as OraCollection

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

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

'create a new OraCollection object from the database
set EnameListNew = OraDatabase.CreateOraObject("ENAMELIST")

'set EnameListNew's element values
EnameListNew(1)  = "Nasser"
EnameListNew(2) =  "Chris"
EnameListNew(3) =  "Gopal"

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'start the AddNew operation and insert the EnameListNew collection
OraDynaset.AddNew
OraDynaset.Fields("dept_id") = 40
OraDynaset.Fields("name") = "DEVELOPMENT"

'set the EnameListNew to enames column
OraDynaset.Fields("enames") = EnameListNew
OraDynaset.Update
```

OraParameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim EnameListNew as OraCollection
```

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

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

'create a new OraCollection object from the database
set EnameListNew = OraDatabase.CreateOraObject("ENAMELIST")

'set EnameListNew's element values
EnameListNew(1)  = "Nasser"
EnameListNew(2) =  "Chris"
EnameListNew(3) =  "Gopal"

'create an input OraParameter object representing EnameList collection bind
'Variable

OraDatabase.Parameters.Add "ENAMES", Null, ORAPARM_INPUT, ORATYPE_VARRAY, _
               "ENAMELIST"

'set the ENAMES parameter value to EnameListNew
OraDatabase.Parameters("ENAMES").Value  = EnameListNew

'execute the insert sql statement
OraDatabase.ExecuteSQL ("insert into department values (40,'DEVELOPMENT', " & _
             ":ENAMES)")
```

Example: Collection with Object Type Elements

The following example illustrates the use of an Oracle collection having elements of object type.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim CourseList as OraCollection
Dim Course as OraObject

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

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

'create a dynaset object from division
set OraDynaset = OraDatabase.CreateDynaset("select * from division", 0&)

'retrieve a Courses column from Division.
'Here Value property of OraField object returns CourseList OraCollection

set CourseList = OraDynaset.Fields("Courses").Value
'retrieve the element value of the CourseList at index 1.
'Here element value is returned as Course OraObject
set Course = CourseList(1)

'retrieve course_no and title attribute of the Course
msgbox Course.course_no
msgbox Course.title

'move to next row
```

```
OraDynaset.MoveNext

'now CourseList object represents collection value for the second row
'and course OraObject 'represents the element value at index 1.
'retrieve course_no and title attribute of the Course.
msgbox Course.course_no
msgbox Course.title
```

Example: Creating a SAFEARRAY Variant from a Collection

The following example illustrates how to get and set a SAFEARRAY `Variant` with an Oracle collection.

Creating SAFEARRAY Variant from a Collection

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameList as OraCollection
Dim EnameArray as Variant

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

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

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'retrieve a Enames column from Department.
'Here Value property of OraField objectreturns EnameList OraCollection
set EnameList = OraDynaset.Fields("Enames").Value

'get the Variant SAFEARRAY from the collection.
EnameArray = EnameList.SafeArray

'display the individual elements of EnameArray
msgbox EnameArray(0)
msgbox EnameArray(1)
msgbox EnameArray(2)
```

Setting SAFEARRAY Variant to the Collection

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim EnameList as OraCollection
Dim EnameArray() As String
ReDim EnameArray(3)

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

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

'create an Empty uninitialized input OraParameter object
'represent EnameList collection bind Variable
OraDatabase.Parameters.Add "ENAMES", Empty, ORAPARM_INPUT, _
                           ORATYPE_VARRAY,"ENAMELIST"
```

```
'get the Empty uninitialized ENAMES parameter value
set EnameList = OraDatabase.Parameters("ENAMES").Value

'initialize the EnameArray
EnameArray(0) = "Nasser"
EnameArray(1) = "Chris"
EnameArray(2) = "Gopal"

'set the EnameArray to EnameList's SafeArray
EnameList.SafeArray = EnameArray

'execute the insert sql statement
OraDatabase.ExecuteSQL ("insert into department " & _
                "values (40,'DEVELOPMENT', :ENAMES)")
```

Example: Creating a Dynaset from a Collection

The following example illustrates how to create a dynaset from an Oracle collection.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim CourseList as OraCollection
Dim Course as OraObject
Dim CourseListDyn as OraDynaset

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

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

'create a dynaset object from division
set OraDynaset = OraDatabase.CreateDynaset("select * from division", 0&)

'retrieve a Courses column from Division. Here Value
'property of OraField object returns CourseList OraCollection
set CourseList = OraDynaset.Fields("Courses").Value

'create a input parameter for CourseList for nested table dynaset
OraDatabase.Parameters.Add "COURSELIST", CourseList, ORAPARM_INPUT, _
                           ORATYPE_TABLE, "COURSELIST"

'create a read only dynaset based on the CourseList.
Set CourseListDyn = OraDatabase.CreateDynaset("select * from THE" & _
          "(select CAST(:COURSELIST AS COURSELIST) from dual)", ORADYN_READONLY)

'dynaset can also be created from Oracle8 collection
'using the following statement, which requires OO4O v8.1.x later

Set CourseListDyn = OraDatabase.CreateDynaset("select * from " & _
           "TABLE(CAST(:COURSELIST AS COURSELIST))", ORADYN_READONLY)

'get the field values of the collection dynaset
msgbox CourseListDyn.Fields("title").Value
msgbox CourseListDyn.Fields("course_no").Value

'move the original dynaset to second row
Oradynaset.MoveNext
```

```
'set the new value of CourseList  collection from the second row of main dynaset
'to the "COURSELIST" parameter
OraDatabase.Parameters("COURSELIST").Value = CourseList

'refresh the collection dynaset. Now the collection dynaset values are refreshed
' with new collection value.
CourseListDyn.Refresh

'get the field values of the collection dynaset
msgbox CourseListDyn.Fields("title").Value
msgbox CourseListDyn.Fields("course_no").Value
```

Example: Collection Iterator

See "Example: OraCollection Iterator" on page 10-88.

See Also:

- OraParameter Object on page 9-50
- CreateOraObject (OraDatabase) Method on page 10-97
- "Instantiating Oracle LOBs, Objects, and Collections" on page 4-2
- "Oracle Collections" on page 4-16

OraConnection Object

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 it 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.

Properties

- Connect Property on page 11-23
- ConnectionOK Property on page 11-26
- DatabaseName Property on page 11-37
- Session Property on page 11-141

Methods

- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235

See Also:

- OraConnections Collection on page 9-66
- OraDatabase Object on page 9-28

OraDatabase Object

Description

An `OraDatabase` interface represents a user session to an Oracle database and provides methods for SQL and PL/SQL execution.

Remarks

An `OraDatabase` interface in Oracle8*i* and higher releases adds additional methods for controlling transactions and creating interfaces representing instances of Oracle object types. Attributes of schema objects can be retrieved using the `Describe` method of the `OraDatabase` interface.

In previous releases, an `OraDatabase` object is created by invoking the `OpenDatabase` method of an `OraSession` interface. The network alias, user name, and password are passed as arguments to this method. In Oracle8*i* and higher releases, invocation of this method results in implicit creation of an `OraServer` object.

As described in the `OraServer` interface description, an `OraDatabase` object can also be created using the `OpenDatabase` method of the `OraServer` interface.

Transaction control methods are available at the `OraDatabase` (user session) level. These methods include:

- `BeginTrans`
- `CommitTrans`
- `Rollback`

For example:

```
MyDatabase.BeginTrans
MyDatabase.ExecuteSQL("delete from emp where empno = 1234")
MyDatabase.CommitTrans
```

> **Note:** If the `AutoCommit` property is set to `True`, transactions are committed automatically, and you do not need to use the transaction control methods.

Properties

- AutoCommit Property on page 11-9
- CacheMaximumSize Property on page 11-18
- CacheOptimalSize Property on page 11-19
- Connect Property on page 11-23
- Connection Property on page 11-25
- ConnectionOK Property on page 11-26
- DatabaseName Property on page 11-37
- LastServerErr Property on page 11-87
- LastServerErrPos Property on page 11-89
- LastServerErrText Property on page 11-90

- Options Property on page 11-114
- Server Property on page 11-137
- Parameters Property on page 11-122
- RDMSVersion Property on page 11-127
- Subscriptions Property on page 11-155

Methods

- BeginTrans Method on page 10-43
- Close Method on page 10-63
- CommitTrans Method on page 10-66
- CreateAQ Method on page 10-79
- CreateCustomDynaset Method on page 10-80
- CreateTempBLOB/CLOB Method on page 10-114
- CreateDynaset Method on page 10-85
- CreateOraObject (OraDatabase) Method on page 10-97
- CreateSQL Method on page 10-111
- Describe Method on page 10-124
- ExecuteSQL Method on page 10-144
- FetchOraRef Method on page 10-149
- LastServerErrReset Method on page 10-189
- MonitorForFailover Method on page 10-194
- Open (OraServer) Method on page 10-210
- RemoveFromPool Method on page 10-232
- Rollback Method on page 10-235

See Also:

- OpenDatabase Method on page 10-212
- OraServer Object on page 9-56
- OraServer Object on page 9-56

OraDynaset Object

Description

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

Remarks

An `OraDynaset` object represents the result set of a SQL `SELECT` query or a PL/SQL cursor variable returned from a stored procedure or function. It is essentially a client-side scrollable and updatable cursor that allows browsing the set of rows generated by the query it executes. It is created by the `CreateDynaset` or `CreateCustomDynaset` method of an `OraDatabase` interface. An `OraDynaset` object can be used to scroll result sets that contain instances of relational and object-relational columns such as `VARRAY`s, nested tables, `Objects`, `REF`s, and LOBs and `BFILE` types.

This object provides transparent mirroring of database operations, such as updates. When data is updated through 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 Oracle Database. This integrity checking is performed only when necessary (such as just before updates occur).

During create and refresh operations, the `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.

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 the `"SELECT ... FOR UPDATE"` statement 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 update rules; for example, 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. Note that the `OpenDatabase` method has an option so that you can decide whether to wait on locks.

Properties

BOF Property on page 11-11
Bookmark Property on page 11-13
BookMarkable Property on page 11-15
CacheBlocks Property on page 11-16
CacheChanged Property on page 11-17
LastModified Property on page 11-86
NoMatch Property on page 11-110
Options Property on page 11-114
RecordCount Property on page 11-128
RowPosition Property on page 11-132

Methods

See Also:

- CreateDynaset Method on page 10-85
- CreateCustomDynaset Method on page 10-80
- OraParameter Object on page 9-50
- RowPosition Property on page 11-132
- Update Method on page 10-257

OraField Object

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 the `Edit` method) or a new record is being added (using the `AddNew` method). Other attempts to assign data to the `Value` property of a field results in an error.

Properties

- OraIDataType Property on page 11-115
- OraMaxDSize Property on page 11-117
- OraMaxSize Property on page 11-118
- OraNullOK Property on page 11-119
- OraPrecision Property on page 11-120
- OraScale Property on page 11-121
- Name Property on page 11-101
- Size Property on page 11-143
- Truncated Property on page 11-163
- Type Property on page 11-164
- Value Property on page 11-173
- XMLAsAttribute Property on page 11-188
- XMLTagName Property on page 11-196

Methods

- AppendChunk Method on page 10-28
- AppendChunkByte Method on page 10-30
- FieldSize Method on page 10-150
- GetChunk Method on page 10-156
- GetChunkByteEx Method on page 10-160
- OriginalName on page 10-217
- ReadChunk Method on page 10-224

See Also:

- AddNew Method on page 10-21
- Edit Method on page 10-134
- OraDynaset Object on page 9-30
- OraFields Collection on page 9-67
- Value Property on page 11-173

OraIntervalDS Object

Description

The `OraIntervalDS` object provides methods for operations on the Oracle `INTERVAL DAY TO SECOND`.This data type represents a period of time in terms of days, hours, minutes, seconds, and nanoseconds.

Remarks

The `OraIntervalDS` object is created by the `OraSession.CreateOraIntervalDS` method or by calling the `Clone` method on an existing `OraIntervalDS` object.

An `OraIntervalDS` object can be bound using the `ServerType ORATYPE_INTERVALDS`. This allows the binding of a value to a parameter associated with an Oracle `INTERVAL DAY TO SECOND` data type in a SQL or PL/SQL statement.

When binding a string associated with an `INTERVAL DAY TO SECOND` data type, the `ServerType` must be specified to be a string type (for example, `ORATYPE_VARCHAR2`, `ORATYPE_STRING`) and the string must be in the format specified by Day HH:MI:SSxFF.

Properties

- Days Property on page 11-42
- Hours Property on page 11-76
- Minutes Property on page 11-97
- Nanonseconds Property on page 11-108
- Seconds Property on page 11-136
- TotalDays Property on page 11-160
- Value (OraIntervalDS) Property on page 11-177

Methods

- Add (OraIntervalDS) Method on page 10-11
- Clone (OraIntervalDS) Method on page 10-55
- Div (OraIntervalDS) Method on page 10-130
- IsEqual (OraIntervalDS) Method on page 10-172
- IsGreater (OraIntervalDS) Method on page 10-177
- IsLess (OraIntervalDS) Method on page 10-182
- Mul (OraIntervalDS) Method on page 10-204
- Neg (OraIntervalDS) Method on page 10-207
- Sub (OraIntervalDS) Method on page 10-241
- ToOraNumber (OraIntervalDS) Method on page 10-247

See Also:

- CreateOraIntervalDS Method on page 10-92
- OraNumber Object on page 9-41
- ServerType Property on page 11-138

OraIntervalYM Object

Description

The `OraIntervalYM` object provides methods for operations on the Oracle `INTERVAL YEAR TO MONTH`.This data type represents a period of time in terms of years and months.

Remarks

The `OraIntervalYM` object is created by the `OraSession.CreateOraIntervalYM` method or by calling the `Clone` method on an existing `OraIntervalYM` object.

An `OraIntervalYM` object can be bound using `ServerType` `ORATYPE_INTERVALYM`. This allows the binding of a value to a parameter associated with an Oracle `INTERVAL YEAR TO MONTH` data type in a SQL or PL/SQL statement.

When binding a string associated with an `INTERVAL YEAR TO MONTH` data type, the `ServerType` must be specified to be a string type (for example, `ORATYPE_VARCHAR2, ORATYPE_STRING`), and the string must be in the format specified by YEARS-MONTHS.

Properties

- Months Property on page 11-100
- Years Property on page 11-200
- TotalYears Property on page 11-161
- Value Property on page 11-173

Methods

- Add (OraIntervalYM) Method on page 10-12
- Clone (OraIntervalYM) Method on page 10-56
- Div (OraIntervalYM) Method on page 10-131
- IsEqual (OraIntervalYM) Method on page 10-173
- IsGreater (OraIntervalYM) Method on page 10-178
- IsLess (OraIntervalYM) Method on page 10-183
- Mul (OraIntervalYM) Method on page 10-205
- Neg (OraIntervalYM) Method on page 10-208
- Sub (OraIntervalYM) Method on page 10-242

See Also:

- CreateOraIntervalYM Method on page 10-94
- OraNumber Object on page 9-41
- ServerType Property on page 11-138

OraMDAttribute Object

Description

Each `OraMDAttribute` object describes an individual attribute. It represents an entry to the attribute table of the `OraMetaData` object. It can be accessed by creating a subscript that uses ordinal integers or by using the name of the attribute.

Remarks

None.

Properties

- Name (OraMDAttribute) Property on page 11-105
- Value (OraMDAttribute) Property on page 11-181
- IsMDObject Property on page 11-78

Methods

None.

Examples

See "Schema Objects Used in OraMetaData Examples" on page A-3 for `OraMetaData` Schema Definitions used in these examples.

Example: Describing a Table

See "Describing a Table Example" on page 10-125.

Example: Describing a User-Defined Type

See "Example: Describing a User-Defined Type" on page 10-126.

Example: Describing Unknown Schema Objects

See "Example: Describing Unknown Schema Objects" on page 10-126.

See Also:

- OraMetaData Object on page 9-39
- Describe Method on page 10-124

OraMetaData Object

Description

The OraMetaData object is returned by invoking the Describe method of the OraDatabase interface. The Describe method takes the name of a schema object, such as the emp table, and returns an OraMetaData object. The OraMetaData object provides methods for dynamically navigating and accessing all the attributes (OraMDAttribute collection) of a schema object described.

An OraMetaData object is a collection of OraMDAttribute objects that represent the description information about a particular schema object in the database. The following table is an example of attributes for a OraMetaData object of type table (ORAMD_TABLE).

Table 9–2 list the ORAMD_TABLE attributes.

Table 9–2 ORAMD_TABLE Attributes

Attribute Name	Value Type	Description
ObjectID	Integer	Object ID.
NumCols	Integer	Number of columns.
ColumnList	OraMetaData	Column list.
IsTyped	Boolean	Is the table typed?
IsTemporary	Boolean	Is the table temporary?
Duration	String	Duration - can be session, transaction, or null.
DBA	Integer	Data block address of the segment header.
TableSpace	Integer	Tablespace in which the table resides.
IsClustered	Boolean	Is the table clustered?
IsPartitioned	Boolean	Is the table partitioned?
IsIndexOnly	Boolean	Is the table index-only?

See Also: "Type (OraMetaData) Property" on page 11-168

Remarks

The OraMetaData object can be visualized as a table with three columns:

- Metadata attribute name
- Metadata attribute value
- Flag specifying whether the Value is another OraMetaData object

The OraMDAttribute objects contained in the OraMetaData object can be accessed by creating a subscript that uses ordinal integers or by using the name of the property. Referencing a subscript that is not in the collection (0 to Count-1) results in the return of a NULL OraMDAttribute object.

Properties

- Count (OraMetaData) Property on page 11-33
- Type (OraMetaData) Property on page 11-168

Methods

- Attribute (OraMetaData) Method on page 10-38

Examples

See "Schema Objects Used in OraMetaData Examples" on page A-3 for `OraMetaData` schema definitions used in these examples.

The following Visual Basic example illustrates a simple use of this facility. It retrieves and displays several attributes of the `emp` table.

```
Set empMD = OraDatabase.Describe("emp")

'Display the name of the Tablespace
msgbox empMD("tablespace")

'Display name, data type, and size of each column in the emp table.
Set empColumnsMD = empMD("Columns")
for I = 1 to empColumns.Count
   Set ColumnMD = empColumnsMD(I)
   MsgBox ColumnMD("Name") & ColumnMD("Data Type") & ColumnMD("Length")
Next I
```

Example: Describing a User-Defined Type

See "Example: Describing a User-Defined Type" on page 10-126

Example: Describing Unknown Schema Objects

See "Example: Describing Unknown Schema Objects" on page 10-126

See Also:

- OraMDAttribute Object on page 9-38
- Describe Method on page 10-124

OraNumber Object

Description

The `OraNumber` interface provides methods for operations on the Oracle `Number` data types. This interface exposes a set of math operations that provide greater precision than is available in some programming environments, such as Visual Basic.

Remarks

The `OraNumber` object can be obtained through the `CreateOraNumber` method of the `OraSession` object or by calling the `Clone` method on an existing `OraNumber`.

All of the methods of the `OraNumber` object that take a numeric argument accept a string, another numeric type, such as a `long` in Visual Basic, or another `OraNumber` object.

> **Note:** If a Visual Basic numeric value (or constant) is used as an argument, it is limited to the maximum precision provided by the language.

The `OraNumber` on which the math operation is called holds the result of the operation (overwriting any previous value). If a `Format` was specified (through the `Format` property), the value of an `OraNumber` must match this format or an error is raised when the `Value` property is accessed.

Properties

- Format (OraNumber) Property on page 11-70
- Value (OraNumber) Property on page 11-182

Methods

- Abs Method on page 10-7
- Add (OraNumber) Method on page 10-13
- ArcCos (OraNumber) Method on page 10-34
- ArcSin (OraNumber) Method on page 10-35
- ArcTan (OraNumber) Method on page 10-36
- ArcTan2 (OraNumber) Method on page 10-37
- Ceil (OraNumber) Method on page 10-47
- Clone (OraNumber) Method on page 10-57
- Cos (OraNumber) Method on page 10-78
- Div (OraNumber) Method on page 10-132
- Exp (OraNumber) Method on page 10-148
- Floor (OraNumber) Method on page 10-153
- HypCos (OraNumber) Method on page 10-168
- HypSin (OraNumber) Method on page 10-169

- HypTan (OraNumber) Method on page 10-170
- IsEqual (OraNumber) Method on page 10-174
- IsGreater (OraNumber) Method on page 10-179
- IsLess (OraNumber) Method on page 10-184
- Ln (OraNumber) Method on page 10-190
- Log (OraNumber) Method on page 10-191
- Mod (OraNumber) Method on page 10-193
- Mul (OraNumber) Method on page 10-206
- Neg (OraNumber) Method on page 10-209
- Power (OraNumber) Method on page 10-219
- Round (OraNumber) Method on page 10-237
- SetPi (OraNumber) Method on page 10-238
- Sin (OraNumber) Method on page 10-239
- Sqrt (OraNumber) Method on page 10-240
- Sub (OraNumber) Method on page 10-243
- Tan (OraNumber) Method on page 10-244
- Trunc (OraNumber) Method on page 10-255

Example

A scientific calculator example program is included as part on the samples installed with Oracle Objects for OLE. See "Demonstration Schema and Code Examples" on page 2-1.

See Also: OraSession Object on page 9-58

OraObject Object

Description

The `OraObject` interface is a representation of an Oracle value instance (non-referenceable object instance or embedded objects). Value instances are instances of an Oracle object type stored in the column of a table or attribute of an another Oracle object instance or element of an Oracle collection.

Remarks

Implicitly an `OraObject` object contains a collection interface for accessing and manipulating (updating and inserting) individual attributes of an value instance. Individual attributes can be accessed by using a subscript or the name of the attribute.

The `OraObject` attribute index starts at `1`. The `Count` property returns the total number of attributes. Each attribute of the underlying value instance is represented as an `OraAttribute` object.

Attribute values are retrieved as variants. The `Variant` type of the attribute depends on the attribute type of the object. Attribute values can be null and can be set to `Null`. For object types `REF`, LOB, and collection, attribute values are returned as corresponding OO4O objects for that type.

The `CreateOraObject` method on the `OraDatabase` object returns the `OraObject` object. The value instance associated with this `OraObject` object is created in the client-side object cache.

For information about executing a member method of a value instance, see "Executing a Member Method of an Oracle Object Instance" on page 4-12.

For information about initializing an `OraObject` object representing a value instance in OO4O or executing a member method of a value instance, see "Instantiating Oracle LOBs, Objects, and Collections" on page 4-2.

Properties

- Count (OraObject/Ref) Property on page 11-34
- IsNull (OraObject) Property on page 11-81
- TypeName (OraObject and OraRef) Property
- Version (OraObject and Ref) Property on page 11-185

Methods

- Clone (OraObject/Ref) Method on page 10-58

Examples

See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3 for schema descriptions used in examples of `OraObject/OraRef` objects.

Example: Accessing Attributes of an OraObject Object

The following example accesses the attributes of the `ADDRESS` value instance in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
```

```
Dim OraDynaset as OraDynaset
Dim Address as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab",0&)

'retrieve a address column from person_tab. Here Value property of OraField
'object returns Address OraObject
set Address = OraDynaset.Fields("Addr").Value

'access the attribute by dot notation
msgbox Address.Street

'access the attribute using '!' notation ( early binding application)
msgbox Address!Street

'access the attribute by index
msgbox Address(1)

'access the attribute by name
msgbox Address("Street")

'access all the attributes of Address OraObject in the dynaset
Do Until OraDynaset.EOF
    For index = 1 To Address.Count
        msgbox Address(index)
    Next Index
OraDynaset.MoveNext
Loop
```

Example: Updating Attributes of an OraObject Object

The following examples modify the attributes of the ADDRESS value instance in the database.

Dynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Address as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab", 0&)

'retrieve a address column from person_tab.
'Here Value property of OraField object returns Address OraObject
```

```
set Address = OraDynaset.Fields("Addr").Value

'start the Edit operation and modify the Street attribute
OraDynaset.Edit
Address.Street  =  "Oracle Parkway"
OraDynaset.Update
```

Parameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim Address as OraObject

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

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

'create an  OraParameter object represent Address object bind Variable
OraDatabase.Parameters.Add "ADDRESS", Empty, ORAPARM_INPUT, ORATYPE_OBJECT, _
                    "ADDRESS"

'get the uninitialized 'Empty' Address object from OraParameter
set Address = OraDatabase.Parameters("ADDRESS").Value

'modify the 'Street' attribute of the Address
Address.Street = "Oracle Parkway"

'execute the sql statement which updates Address in the person_tab
OraDatabase.ExecuteSQL ("update person_tab set addr = :ADDRESS where age = 40")
```

Example: Inserting an OraObject Object

The following examples insert a new field (value instance) called `ADDRESS` in the database.

Dynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim AddressNew as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab", 0&)

' create a new Address object in OO4O
set AddressNew = OraDatabase.CreateOraObject("ADDRESS")

'initialize the Address object attribute to new value
AddressNew.Street = "Oracle Parkway"
AddressNew.State = "CA"

'start the dynaset AddNew operation and set the Address field to new address
```

```
' value
OraDynaset.Addnew
OraDynaset.Fields("ADDR").Value = AddressNew
OraDynaset.Update
```

OraParameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim AddressNew as OraObject

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

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

'create an  OraParameter object represent Address object bind Variable
OraDatabase.Parameters.Add "ADDRESS", Null, ORAPARM_INPUT, ORATYPE_OBJECT, _
                    "ADDRESS"

' create a new Address object in OO4O
set AddressNew = OraDatabase.CreateObject("ADDRESS")

'initialize the Address object attribute to new value
AddressNew.Street = "Oracle Parkway"
AddressNew.State = "CA"

'set the Address to ADDRESS parameter
Oradatabase.Parameters("ADDRESS").Value = AddressNew

'execute the sql statement which updates Address in the person_tab
OraDatabase.ExecuteSQL ("insert into person_tab values (30,'Eric',:ADDRESS))
```

See Also:

- "Oracle Object Data Types" on page 4-10 for information on support of Oracle object-relational features
- OraParameter Object on page 9-50
- OraParamArray Object on page 9-47
- OraRef Object on page 9-52
- OraAttribute Object on page 9-7
- CreateOraObject (OraDatabase) Method on page 10-97
- "Executing a Member Method of an Oracle Object Instance" on page 4-12

OraParamArray Object

Description

An `OraParamArray` object represents an *array* type bind variable in a SQL statement or PL/SQL block, as opposed to a *scalar* type bind variable represented by the `OraParameter` object.

Remarks

`OraParamArray` 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.

Implicitly an `OraParamArray` object contains an OLE automation collection interface for accessing and manipulating individual elements of an array. Individual elements can be accessed using a subscript or the `Get_Value` method. Individual elements can be modified by using a subscript or the `Put_Value` method.

Element values are retrieved as `Variant` types. The `Variant` type of the element depends on the `ServerType` of the `OraParamArray` object. Element values can be null and can be set to `Null`. For elements of type objects and `REF`s, element values are returned as corresponding OO4O objects for that type.

You can automatically bind a parameter to SQL and PL/SQL statements of other objects (as noted in the objects descriptions) by using the name of the parameter as a placeholder in the SQL or PL/SQL statement. Using parameters can simplify dynamic queries and increase program performance. Parameters are bound to SQL statements and PL/SQL blocks before execution.

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 (through `CreateDynaset`, `ExecuteSQL`, or `CreateSQL` methods). 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 does not apply to that particular SQL statement or PL/SQL block) are noted and no attempt is made to bind them again if the SQL statement or PL/SQL block is reexecuted but does not change.

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

Properties

- ArraySize Property on page 11-8
- LastErrorText Property on page 11-85
- MinimumSize Property on page 11-93
- Name Property on page 11-101
- ServerType Property on page 11-138
- Status Property on page 11-152
- Type Property on page 11-164

Methods

- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41
- Get_Value Method on page 10-167
- Put_Value Method on page 10-220

Example

Example: Using OraParamArrays with SQL Statements

The following example shows how to use the `OraParamArray` object with SQL statements:

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraSqlStmt As OraSQLStmt
Dim PartNoArray As OraParamArray
Dim DescArray As OraParamArray
Dim I As Integer

'Test case for inserting/updating/deleting multiple rows using parameter

' arrays with SQL statements
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase("exampledb", "scott/tiger",  0&)

'Create table
OraDatabase.ExecuteSQL ("create table part_nos(partno number," & _
                        "description char(50), primary key(partno))")
OraDatabase.Parameters.AddTable "PARTNO", ORAPARM_INPUT, ORATYPE_NUMBER, 10, 22
OraDatabase.Parameters.AddTable "DESCRIPTION", ORAPARM_INPUT, _
                          ORATYPE_CHAR, 10, 50
Set PartNoArray = OraDatabase.Parameters("PARTNO")
Set DescArray = OraDatabase.Parameters("DESCRIPTION")

 'Initialize arrays
For I = 0 To 9
  achar = "Description" + Str(I)
  PartNoArray(I) = 1000 + I
  DescArray(I) = achar
Next I
Set OraSqlStmt = OraDatabase.CreateSql("insert into
part_nos(partno, description) values(:PARTNO,:DESCRIPTION)", 0&)

'Update the newly created part_nos table
For I = 0 To 9
  achar = "Description" + Str(1000 + I)
  DescArray(I) = achar
Next I

'Update table
Set OraSqlStmt = OraDatabase.CreateSql("update part_nos set DESCRIPTION" & _
                          "=:DESCRIPTION where PARTNO = :PARTNO", 0&)

'Deleting rows
Set OraSqlStmt = OraDatabase.CreateSql("delete  from  part_nos where" & _
                          "DESCRIPTION=: Description ", 0&)
```

```
'Drop the table
OraDatabase.ExecuteSQL ("drop table part_nos")
```

Example: Using OraParamArrays with PL/SQL

The following is an example using `OraParamArray` objects with PL/SQL. The `Employee` PL/SQL package can be set up with the `ORAEXAMP.SQL` script. See "Demonstration Schema and Code Examples" on page 2-1.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim EmpnoArray As OraParamArray
Dim EnameArray As OraParamArray

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase("exampledb", "scott/tiger", 0&)
OraDatabase.Parameters.Add "ArraySize", 3, ORAPARM_INPUT
OraDatabase.Parameters.AddTable "EMPNOS", ORAPARM_INPUT, ORATYPE_NUMBER,3, 22
OraDatabase.Parameters.AddTable "ENAMES", ORAPARM_OUTPUT, _
                                ORATYPE_VARCHAR2, 3, 10
Set EmpnoArray = OraDatabase.Parameters("EMPNOS")
Set EnameArray = OraDatabase.Parameters("ENAMES")

'Initialize the newly created input parameter table EMPNOS
EmpnoArray(0) = 7698
EmpnoArray(1) = 7782
EmpnoArray(2) = 7654

'Execute the PLSQL package
OraDatabase.ExecuteSQL ("Begin Employee.GetEmpNamesInArray(:ArraySize," & _
                        ":EMPNOS, :ENAMES); End;")
'Print out Enames
MsgBox EnameArray(0)
MsgBox EnameArray(1)
MsgBox EnameArray(2)
```

See Also:

- OraParameters Collection on page 9-68
- OraParameter Object on page 9-50

OraParameter Object

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 object descriptions), by using the parameter name as a placeholder in the SQL or PL/SQL statement. Using 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. Therefore, all parameters are available to any SQL statement or PL/SQL block executed within the database (through the CreateDynaset or ExecuteSQL methods).

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 does not apply to that particular SQL statement or PL/SQL block), are noted and no attempt is made to bind them again if the SQL statement or PL/SQL block is reexecuted but does not change.

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

By default, the maximum size of the ORAPARM_OUTPUT variable for ServerType CHAR and VARCHAR2 is set to 127 bytes. Use the MinimumSize property to change this value. The minimum size of an ORAPARM_OUTPUT variable for CHAR, VARCHAR2, and ORATYPE_RAW_BIN must always be greater than the size of the expected data from the database column.

ServerType ORATYPE_RAW_BIN is used when binding to Oracle Raw columns. A byte array is used to put or get values. The maximum allowable size of ORATYPE_RAW_BIN bind buffers is 2000 bytes when bound to a column of a table, 32 KB when bound to a stored procedure. For example code, see the samples in the *ORACLE_BASE**ORACLE_HOME*\OO4O\VB\Raw directory.

Properties

- DynasetOption Property on page 11-50
- MinimumSize Property on page 11-93
- Name Property on page 11-101
- ServerType Property on page 11-138
- Status Property on page 11-152
- Type Property on page 11-164
- Value Property on page 11-173

Methods

- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41
- DynasetCacheParams Method on page 10-133

See Also:

- CreateDynaset Method on page 10-85
- DynasetOption Property on page 11-50
- ExecuteSQL Method on page 10-144
- OraDatabase Object on page 9-28
- OraParameters Collection on page 9-68

OraRef Object

Description

The OraRef interface represents an Oracle REF (reference) as well as a referenceable object (standalone instance).

Remarks

An Oracle REF is an identifier to a referenceable object. Referenceable objects are stored in rows of an object table. By pinning a REF object, referenceable objects are fetched to the client side. An OraRef object implicitly pins the underlying REF when the attributes of a referenceable object are accessed for the first time. The OraRef also encapsulates the functionality for an object navigational operation utilizing the Complex Object Retrieval Capability (COR).

Attributes of a referenceable object represented by the OraRef object are accessed in the same manner as attributes of an value instance represented by the OraObject interface. When pinned, OraRef contains an OraObject interface through the containment mechanism in COM. At run time, the OraRef interface can be typecast to the OraObject interface.

OraRef provides methods for update and delete operations on a referenceable object, independent of the context from which they originated, such as dynasets, parameters, and so on.

An object-level lock should be obtained before modifying the attributes of a referenceable object. This is done though the Edit method of the OraRefobject.

The CreateOraObject method on the OraDatabase object creates a new referenceable object in the database and returns information associated with the OraRef Object. The CreateOraObject and Update methods pair inserts a new referenceable object in the database.

For information about initializing an OraRef object representing a referenceable object in OO4O or executing a member method of a referenceable object, see "Instantiating Oracle LOBs, Objects, and Collections" on page 4-2.

Properties

- Count (OraObject/Ref) Property on page 11-34
- EditOption (OraRef) Property on page 11-52
- HexValue (OraRef) Property on page 11-73
- IsRefNull (OraRef) Property on page 11-84
- PinOption (OraRef) Property on page 11-123
- TableName (OraRef) Property on page 11-156
- TypeName (OraObject and OraRef) Property
- Version (OraObject and Ref) Property on page 11-185

Methods

- CancelEdit (OraRef) Method on page 10-46
- Clone (OraObject/Ref) Method on page 10-58

- Delete (OraRef) Method on page 10-120
- Edit (OraRef) Method on page 10-136
- Refresh (OraRef) Method on page 10-228
- Update (OraRef) Method on page 10-259

Examples

Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3 for schema descriptions used in examples of `OraObject/OraRef`.

Example: Pinning Ref Values

The following example pins the attributes of the `PERSON` referenceable object in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)

'retrieve a aperson column from customers. Here Value property of
' OraField object returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'access the attribute of person. This operation pins the Person ref
'value and fetches the Person referenceable object to the client.
msgbox Person.Name
```

Example: Accessing Attribute Values

The following example accesses the attributes of the `PERSON` referenceable object in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef
Dim Address as OraObject

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)
```

```
'retrieve a aperson column from customers. Here Value property of OraField
'object returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'access the attribute by dot notation.
msgbox Person.Name

'access the attribute using '!' notation ( early binding application)
msgbox Person!Name

'access the attribute by index
msgbox Person(1)

'access the attribute by name
msgbox Person("Name")

'access Addr attribute . This returns Address OraObject.
set Address = Person.Addr
```

Example: Updating Attribute Values

The following example updates the attributes of the PERSON referenceable object in the database.

Dynaset Example

See "Updating Attribute Values: Dynaset Example" on page 10-259.

Parameter Example

See "Updating Attribute Values: Parameter Example" on page 10-259.

Example: Inserting Referenceable Objects

The following example inserts the new PERSON referenceable object in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim Person  as OraRef

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

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

'CreateOraObject   creates a new referenceable object in the PERSON_TAB object
'table and returns associated OraRef
set Person = OraDatabase.CreateOraObject("PERSON","PERSON_TAB")

'modify the attributes of Person
Person.Name = "Eric"
Person.Age = 35

'Update method inserts modified referenceable object in the PERSON_TAB.
Person.Update
```

See Also:

- OraObject Object on page 9-43
- OraParameter Object on page 9-50
- OraParamArray Object on page 9-47

OraServer Object

Description

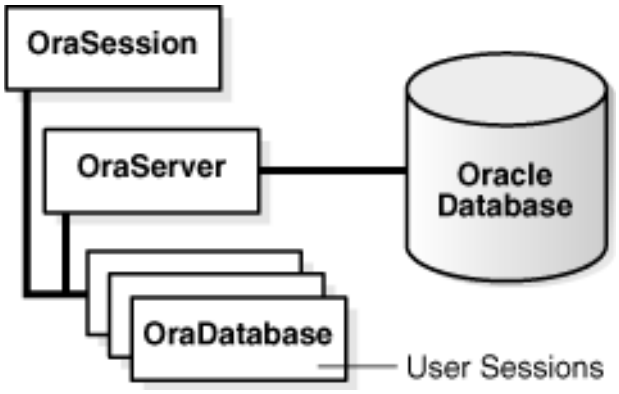
The `OraServer` interface represents a physical network connection to an Oracle database.

Remarks

The `OraServer` interface exposes the connection multiplexing feature provided in the Oracle Call Interface. After an `OraServer` object is created, multiple user sessions (`OraDatabase`) can be attached to it by invoking the `OpenDatabase` method. This feature is particularly useful for application components, such as Internet Information Server (IIS), that use Oracle Objects for OLE in n-tier distributed environments. The use of connection multiplexing when accessing Oracle databases with a large number of user sessions active can help reduce server processing and resource requirements while improving the database scalability.

As illustrated in Figure 9–1, the `OraServer` interface contains a connection to an Oracle database and provides a method (`OpenDatabase`) for creating user sessions (`OraDatabase` objects) on the database connection it contains.

Figure 9–1 OraServer to Oracle Database Relationship



Properties

- Name Property on page 11-101
- Session Property on page 11-141
- Databases Property on page 11-39

Methods

- ChangePassword (OraServer) Method on page 10-48
- OpenDatabase Method on page 10-212
- Open (OraServer) Method on page 10-210

See Also:

- OraConnection Object on page 9-27
- OraDatabase Object on page 9-28
- OraClient Object on page 9-18
- OraDynaset Object on page 9-30

OraSession Object

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 highest level object for an application. `OraSession` and `OraServer` objects are the only objects created by the `CreateObject` Visual Basic or Visual Basic for Applications APIs and not by an Oracle Objects for OLE method.

Properties

- Client Property on page 11-22
- Connections Property on page 11-27
- LastServerErr Property on page 11-87
- LastServerErrText Property on page 11-90
- Name Property on page 11-101
- OIPVersionNumber Property on page 11-113
- DbPoolCurrentSize Property on page 11-43
- DbPoolInitialSize Property on page 11-44
- DbPoolMaxSize Property on page 11-45

Methods

- BeginTrans Method on page 10-43
- ChangePassword (OraSession) Method on page 10-50
- CommitTrans Method on page 10-66
- ConnectSession Method on page 10-69
- CreateDatabasePool Method on page 10-83
- CreateNamedSession Method on page 10-90
- CreateOraIntervalDS Method on page 10-92
- CreateOraIntervalYM Method on page 10-94
- CreateOraNumber Method on page 10-96
- CreateOraTimeStamp Method on page 10-100
- CreateOraTimeStampTZ Method on page 10-102
- DestroyDatabasePool Method on page 10-128
- GetDatabaseFromPool Method on page 10-155
- OpenDatabase Method on page 10-212

- LastServerErrReset Method on page 10-189
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235

Examples

The following code fragments show how to create an `OraSession` object:

```
Dim oo4oSession as Object
Set oo4oSession = CreateObject("OracleInProcServer.XOraSession")
```

or

```
Dim oo4oSession as New OraSessionClass
```

or

```
Dim oo4oSession as OraSession
Set oo4oSession = New OraSessionClass
```

See Also:

- OraClient Object on page 9-18
- OraConnection Object on page 9-27
- OraDatabase Object on page 9-28
- OraDynaset Object on page 9-30
- OraServer Object on page 9-56

OraSQLStmt Object

Description

An `OraSQLStmt` object represents a single SQL statement. Use the `CreateSQL` method to create the `OraSQLStmt` object from an `OraDatabase` object.

During create and refresh operations, `OraSQLStmt` 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 improve the performance of SQL statement execution without parsing the SQL statement again.

Properties

- Connection Property on page 11-25
- Database Property on page 11-36
- Options Property on page 11-114
- RecordCount Property on page 11-128
- Session Property on page 11-141
- SQL Property on page 11-150
- NonBlockingState Property on page 11-111

Methods

- Refresh Method on page 10-225
- Cancel Method on page 10-45
- Close Method on page 10-63

See Also:

- CreateSQL Method on page 10-111
- OraParameter Object on page 9-50
- "Asynchronous Processing" on page 3-16

OraSubscription Object

Description

An `OraSubscription` object that represents the subscription to a database event.

Remarks

`OraSubscription` objects are created, accessed, and removed indirectly through the `OraSubscriptions` collection of an `OraDatabase` object. Each subscription has a name that associates with an Oracle database event.

The `OraSubscriptions` collection is part of the `OraDatabase` object.

Properties

- Name Property on page 11-101

Methods

- Register Method on page 10-229
- Unregister Method on page 10-256

See Also:

- OraDatabase Object on page 9-28
- OraSubscriptions Collection on page 9-70
- "Database Events" on page 4-22

OraTimeStamp Object

Description

The OraTimeStamp object represents the Oracle TIMESTAMP and Oracle TIMESTAMP WITH LOCAL TIME ZONE data types and provides methods for operations on these two Oracle data types. The OraTimeStamp represents a date-time value that stores the following information: year, day, hour, minute, second, and nanosecond.

Remarks

The OraTimeStamp object is created by the OraSession.OraCreateTimeStamp method or by calling the Clone method on an existing OraTimeStamp object.

An OraTimeStamp object can be bound using ServerType ORATYPE_TIMESTAMP or ORATYPE_TIMESTAMPLTZ. This allows the binding of a value to a parameter associated with an Oracle TIMESTAMP or an Oracle TIMESTAMP WITH LOCAL TIME ZONE data type in a SQL or PL/SQL statement respectively.

When binding a string associated with a TIMESTAMP or a TIMESTAMP WITH LOCAL TIME ZONE data types, the ServerType must be specified to be a string type (for example, ORATYPE_VARCHAR2, ORATYPE_STRING) and the string must be in the format specified by the NLS_TIMESTAMP_FORMAT.

Properties

- Day (OraTimeStamp) Property on page 11-40
- Format (OraTimeStamp) Property on page 11-71
- Hour (OraTimeStamp) Property on page 11-74
- Minute (OraTimeStamp) Property on page 11-95
- Month (OraTimeStamp) Property on page 11-98
- Nanosecond(OraTimeStamp) Property on page 11-106
- Second (OraTimeStamp) Property on page 11-134
- Value (OraTimeStamp) Property on page 11-183
- Year (OraTimeStamp) Property on page 11-198

Methods

- AddIntervalDS Method on page 10-17
- AddIntervalYM Method on page 10-19
- Clone (OraTimeStamp) Method on page 10-61
- IsEqual (OraTimeStamp) Method on page 10-175
- IsGreater (OraTimeStamp) Method on page 10-180
- IsLess (OraTimeStamp) Method on page 10-185
- ToDate Method on page 10-245
- ToOraTimeStampTZ Method on page 10-250

See Also:

- CreateOraTimeStamp Method on page 10-100
- OraNumber Object on page 9-41
- ServerType Property on page 11-138

OraTimeStampTZ Object

Description

The OraTimeStampTZ object represents an Oracle TIMESTAMP WITH TIME ZONE data type and provides methods for operations on this Oracle data type. The OraTimeStampTZ represents a date-time value in a specific time zone that stores the following information: year, day, hour, minute, second, nanosecond, and the time zone.

Remarks

The OraTimeStampTZ object is created by the OraSession.OraCreateTimeStampTZ method or by calling the Clone method on an existing OraTimeStampTZ object.

An OraTimeStampTZ object can be bound using ServerType ORATYPE_TIMESTAMPTZ. This allows the binding of a value to a parameter associated with an Oracle TIMESTAMP WITH TIME ZONE data type in a SQL or PL/SQL statement.

When binding a string associated with an TIMESTAMP WITH TIME ZONE data type, the ServerType must be specified to be a string type (for example, ORATYPE_VARCHAR2, ORATYPE_STRING) and the string must be in the format specified by NLS_TIMESTAMP_TZ_FORMAT.

Properties

- Day (OraTimeStampTZ) Property on page 11-41
- Format (OraTimeStampTZ) Property on page 11-72
- Hour (OraTimeStampTZ) Property on page 11-75
- Minute (OraTimeStampTZ) Property on page 11-96
- Month (OraTimeStampTZ) Property on page 11-99
- Nanonsecond (OraTimeStampTZ) Property on page 11-107
- Second (OraTimeStampTZ) Property on page 11-135
- TimeZone (OraTimeStampTZ) Property on page 11-158
- Value (OraTimeStampTZ) Property on page 11-184
- Year (OraTimeStampTZ) Property on page 11-199

Methods

- AddIntervalDS Method on page 10-17
- AddIntervalYM Method on page 10-19
- Clone (OraTimeStampTZ) Method on page 10-62
- IsEqual (OraTimeStampTZ) Method on page 10-176
- IsGreater (OraTimeStampTZ) Method on page 10-181
- IsLess (OraTimeStampTZ) Method on page 10-186
- ToDate Method on page 10-245
- ToOraTimeStamp Method on page 10-248

- ToOraTimeStampLTZ Method on page 10-249
- ToUniversalTime Method on page 10-251

See Also:

- CreateOraTimeStampTZ Method on page 10-102
- OraNumber Object on page 9-41
- ServerType Property on page 11-138

OraConnections Collection

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 creating a subscript (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 is not within the collection (`0` to `Count-1`) results in the return of a `NULL` `OraConnection` object.

Properties

- Count Property on page 11-31

Methods

None.

See Also: OraConnection Object on page 9-27

OraFields Collection

Description

The `OraFields` collection maintains a list of the `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 creating a subscript (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 is not within the collection (`0` to `Count-1`) results in the return of a null `OraField` object.

Properties

- Count Property on page 11-31

Methods

- OriginalItem Method on page 10-215

See Also: OraField Object on page 9-33

OraParameters Collection

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 creating a subscript (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 is not 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.

Properties

- Count Property on page 11-31

Methods

- Add Method on page 10-8
- AddTable Method on page 10-23
- Remove Method on page 10-230

See Also:

- OraParameter Object on page 9-50
- OraParamArray Object on page 9-47

OraSessions Collection

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 creating a subscript (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 is not within the collection (`0` to `Count-1`) results in the return of a null `OraSession` object.

Properties

- Count Property on page 11-31

Methods

None.

See Also: OraSession Object on page 9-58

OraSubscriptions Collection

Description

The `OraSubscriptions` collection maintains a list of `OraSubscription` objects, which represent the subscription to a database event. Unlike the other collection objects, this list is modifiable; you can add to and remove from the collection.

Remarks

You can access the `OraSubscription` objects in this collection by creating a subscript (using ordinal integers) or by using the name the object was given at its creation. You can obtain the number of `OraSubscription` objects in the collection by using the `Count` property. Referencing a subscript that is not within the collection (`0` to `Count-1`) results in the return of a null `OraSubscription` object.

In addition to accessing the `OraSubscription` objects of the collection, you can use the collection to create and destroy subscriptions by using the `Add` and `Remove` methods, respectively.

Properties

- Count Property on page 11-31

Methods

- Add (OraSubscriptions Collection) Method on page 10-14
- Remove (OraSubscriptions Collection) Method on page 10-231

See Also:

- OraSubscription Object on page 9-61
- OraDatabase Object on page 9-28
- "Database Events" on page 4-22

10

Server Methods

This chapter describes the Oracle Objects for OLE Server methods.

For an introduction to OO4O server objects, see "Oracle Objects for OLE In-Process Automation Server" on page 1-2.

This chapter contains these topics:

Server Methods: A to B

- ArcSin (OraNumber) Method
- ArcTan (OraNumber) Method
- ArcTan2 (OraNumber) Method
- Attribute (OraMetaData) Method
- AutoBindDisable Method
- AutoBindEnable Method
- BeginTrans Method

Server Methods: C

- Cancel Method
- CancelEdit (OraRef) Method
- Ceil (OraNumber) Method
- ChangePassword (OraServer) Method
- ChangePassword (OraSession) Method
- Clone Method
- Clone (OraLOB/BFILE) Method
- Clone (OraCollection) Method
- Clone (OraIntervalDS) Method
- Clone (OraIntervalYM) Method
- Clone (OraNumber) Method
- Clone (OraObject/Ref) Method
- Clone (OraTimeStamp) Method
- Clone (OraTimeStampTZ) Method
- Close Method
- Close (OraBFILE) Method
- CloseAll (OraBFILE) Method
- CommitTrans Method
- Compare (OraLOB) Method
- ConnectSession Method
- CopyToClipboard Method
- Copy (OraLOB) Method
- CopyFromFile (OraLOB) Method
- CopyFromBFILE (OraLOB) Method
- CopyToFile (OraLOB/BFILE) Method
- Cos (OraNumber) Method
- CreateAQ Method
- CreateCustomDynaset Method
- CreateDatabasePool Method

- CreateDynaset Method
- CreateIterator Method
- CreateNamedSession Method
- CreateOraIntervalDS Method
- CreateOraIntervalYM Method
- CreateOraNumber Method
- CreateOraObject (OraDatabase) Method
- CreateOraTimeStamp Method
- CreateOraTimeStampTZ Method
- CreatePLSQLCustomDynaset Method
- CreatePLSQLDynaset Method
- CreateSession Method
- CreateSQL Method
- CreateTempBLOB/CLOB Method

Server Methods: D to H

- Delete Method
- Delete (OraCollection) Method
- Delete (OraRef) Method
- DeleteIterator Method
- Dequeue (OraAQ) Method
- Describe Method
- DestroyDatabasePool Method
- DisableBuffering (OraLOB) Method
- Div (OraIntervalDS) Method
- Div (OraIntervalYM) Method
- Div (OraNumber) Method
- DynasetCacheParams Method
- Edit Method
- Edit (OraRef) Method
- ElementValue Method
- EnableBuffering (OraLOB) Method
- Enqueue (OraAQ) Method
- Erase (OraLOB) Method
- ExecuteSQL Method
- Exist (OraCollection) Method
- Exp (OraNumber) Method
- FetchOraRef Method

- FieldSize Method
- FindFirst, FindLast, FindNext, and FindPrevious Methods
- Floor (OraNumber) Method
- FlushBuffer (OraLOB) Method
- GetDatabaseFromPool Method
- GetChunk Method
- GetChunkByte Method
- GetChunkByteEx Method
- GetXML Method
- GetXMLToFile Method
- GetRows Method
- Get_Value Method
- HypCos (OraNumber) Method
- HypSin (OraNumber) Method
- HypTan (OraNumber) Method

Server Methods: I to L

- InitIterator Method
- IsEqual (OraIntervalDS) Method
- IsEqual (OraIntervalYM) Method
- IsEqual (OraNumber) Method
- IsEqual (OraTimeStamp) Method
- IsEqual (OraTimeStampTZ) Method
- IsGreater (OraIntervalDS) Method
- IsGreater (OraIntervalYM) Method
- IsGreater (OraNumber) Method
- IsGreater (OraTimeStamp) Method
- IsGreater (OraTimeStampTZ) Method
- IsLess (OraIntervalDS) Method
- IsLess (OraIntervalYM) Method
- IsLess (OraTimeStamp) Method
- IsLess (OraTimeStampTZ) Method
- IterNext Method
- IterPrev Method
- LastServerErrReset Method
- Ln (OraNumber) Method
- Log (OraNumber) Method

Server Methods: M to S

- MatchPos (OraLOB/BFILE) Method
- Mod (OraNumber) Method
- MonitorForFailover Method
- MonitorStart (OraAQ) Method
- MonitorStop (OraAQ) Method
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods
- MovePreviousn, MoveNextn, MoveRel, and MoveTo Methods
- Mul (OraIntervalDS) Method
- Mul (OraIntervalYM) Method
- Mul (OraNumber) Method
- Neg (OraIntervalDS) Method
- Neg (OraIntervalYM) Method
- Neg (OraNumber) Method
- Open (OraServer) Method
- Open (OraBFILE) Method
- OpenDatabase Method
- OriginalItem Method
- OriginalName
- Power (OraNumber) Method
- Put_Value Method
- Read (OraLOB/BFILE) Method
- ReadChunk Method
- Refresh Method
- Refresh (OraRef) Method
- Register Method
- Remove Method
- Remove (OraSubscriptions Collection) Method
- RemoveFromPool Method
- ResetTrans Method
- Rollback Method
- Round (OraNumber) Method
- SetPi (OraNumber) Method
- Sin (OraNumber) Method
- Sqrt (OraNumber) Method
- Sub (OraIntervalDS) Method
- Sub (OraIntervalYM) Method

- Sub (OraNumber) Method

Server Methods: T to Z

- Tan (OraNumber) Method
- ToDate Method
- ToOraNumber (OraIntervalDS) Method
- ToOraTimeStamp Method
- ToOraTimeStampLTZ Method
- ToOraTimeStampTZ Method
- ToUniversalTime Method
- Trim (OraCollection) Method
- Trim (OraLOB) Method
- Trunc (OraNumber) Method
- Unregister Method
- Update Method
- Update (OraRef) Method
- Write (OraLOB) Method

Abs Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the absolute value of an `OraNumber` object.

Usage

```
OraNumber.Abs
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Add Method

Applies To

OraParameters Collection on page 9-68

Description

Adds a parameter to the `OraParameters` collection.

Usage

```
oraparameters.Add Name, Value, IOType, ServerType, ObjectName
```

Arguments

The arguments for the method are:

Arguments	Description
Name	The name of the parameter to be added to the parameters collection. This name is issued 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; 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.
ServerType	Specifies Oracle Database type to which this parameter is to be bound. This is required when binding to `BLOB`, `CLOB`, `BFILE`, `OBJECT`, `REF`, `NESTED TABLE`, or `VARRAY`. For a list of possible values, see the `OraParameter` "ServerType Property" on page 11-138.
ObjectName	A case-sensitive string containing the name of the `Object`. This is only required if *ServerType* is `ORATYPE_OBJECT`, `ORATYPE_VARRAY`, or `ORATYPE_TABLE`. *ServerType* is required for `ORATYPE_REF` when the `REF` is used in PL/SQL.

IOType Settings

The *IOType* settings are:

Settings	Values	Description
`ORAPARM_INPUT`	1	Used for input variables only
`ORAPARM_OUTPUT`	2	Used for output variables only
`ORAPARM_BOTH`	3	Used for variables that are both input and output

These values can be found in the `oraconst.txt` file.

By default, the maximum size of the `ORAPARM_OUTPUT` variable for `ServerType` `VAR`, `VARCHAR2`, and `ORATYPE_RAW_BIN` is set to 128 bytes. Use the `MinimumSize` property to change this value. The minimum size of an `ORAPARM_OUTPUT` variable for `VAR` and `VARCHAR2` must always be greater than the size of the expected data from the database column.

Verify that this value is correct. If you set an incorrect option, such as ORAPARM_BOTH for the IN stored procedure parameter type, this can result in errors. ORAPARM_BOTH is for IN and OUT parameters only. It is not used against one stored procedure that has an IN parameter and another that has an OUT parameter. For this case, use two parameters. Errors caused this way are rare, if there is a parameter-related error, verify that the *IOType* is correct.

The *Value* argument can be an Oracle Database 10*g* object, such as an OraBLOB. Note that a copy of the object is made at that point in time and the Value property must be accessed to obtain a new object that refers to the value of the parameter. For example, if *IOType* is ORATYPE_BOTH and an OraBLOB obtained from a dynaset is passed in as the input value, the Parameter Value property needs to be accessed one time after the SQL has been executed to obtain the newly updated output value of the parameter. The object is obtained from the parameter in the same manner as from a dynaset.

The Value property always refers to the latest value of the parameter. The Visual Basic value Null can also be passed as a value. The Visual Basic EMPTY value can be used for BLOB and CLOB data types to mean an empty LOB, and the EMPTY value can be used for OBJECT, VARRAY, and NESTED TABLE data types to mean an object whose attributes are all Null.

Remarks

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

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

The ORATYPE_RAW_BIN ServerType value is used when binding to Oracle Raw columns. A byte array is used to Put or Get values. The maximum allowable size of an ORATYPE_RAW_BIN bind buffers is 2000 bytes when bound to a column of a table and 32 KB when bound to a stored procedure. For example code, see the samples in the *ORACLE_BASE**ORACLE_HOME*\OO4O\VB\Raw directory.

Examples

This example demonstrates using the Add and Remove parameter methods, the ServerType parameter property, and 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
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase

'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.ExecuteSQL ("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.ExecuteSQL ("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
```

See Also:

- OraParameter Object on page 9-50
- Remove Method on page 10-230
- ServerType Property on page 11-138

Add (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Adds an argument to the `OraIntervalDS` object.

Usage

```
OraIntervalDS.Add operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object to be added.

Remarks

The result of the operation is stored in an `OraIntervalDS` object, overwriting any previous value. There is no return value.

If *operand* is a `Variant` of type `String`, it must be in the following format: [+/-]Day HH:MI:SSxFF.

If *operand* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` object represents.

Examples

```
Dim oraIDS as OraIntervalDS

'Create an OraIntervalDS using a string which represents
'1 day and 12 hours
Set oraIDS = oo4oSession.CreateOraIntervalDS("1 12:0:0.0")

'Add an interval using a string, which represents 2 days
'and 12 hours, to oraIDS.
'The resulting oraIDS is an interval which represents 4 days
oraIDS.Add "2 12:0:0.0"
```

See Also: "CreateOraIntervalDS Method" on page 10-92

Add (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Adds an argument to the `OraIntervalYM` object.

Usage

```
OraIntervalYMObj.Add operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object to be added.

Remarks

The result of the operation is stored in the `OraIntervalYM` object, overwriting any previous value. There is no return value.

If *operand* is a `Variant` of type `String`, it must be in the following format: [+/-]YEARS-MONTHS.

If *operand* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

Examples

```
Dim oraIYM as OraIntervalYM

'Create an OraIntervalYM using a string which represents 1 year and 6 months
Set oraIYM = oo4oSession.CreateOraIntervalYM("1-6")

'Add an interval using a string, which represents 2 years
'and 6 months, to oraIYM.
'The resulting oraIYM is an interval which represents 4 years
oraIYM.Add "2-6"
```

See Also: CreateOraIntervalYM Method on page 10-94

Add (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Adds a numeric argument to the `OraNumber` object.

Usage

```
OraNumber.Add operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber` object, or a numeric value.

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

Add (OraSubscriptions Collection) Method

Applies To

OraSubscriptions Collection on page 9-70

Description

Adds a subscription to the `OraSubscriptions` collection.

Usage

```
orasubscriptions.Add Name, DbeventsHdl, Ctx
```

Arguments

The arguments for the method are:

Variants	Description
[in] *Name*	The database event of interest. The appropriate event trigger and AQ queue must be set up prior to this. *Name* refers to the subscription name in the form of the string `'SCHEMA.QUEUE'` if the registration is for a single consumer queue and `'SCHEMA.QUEUE:CONSUMER_NAME'` if the registration is for a multiple consumer queue. The *Name* string should be in uppercase.
[in] *DbeventsHdl*	The database event handler. An `IDispatch` interface implementing the `NotifyDBEvents` method, which is invoked when the database event of interest is fired.
[in] *Ctx*	Context-specific information that the application wants passed to the `NotifyDbEvents` method when it is invoked.

Remarks

To register for subscription of a database event, the name identifying the subscription of interest and the name of the `dbevent` handler that handles the event must be passed in when the `Add` method is called. The queues and event triggers necessary to support the database event must be set up before the subscriptions can be fired.

The `dbevent` handler should be an automation object that implements the `NotifyDBEvents` method.

NotifyDBEvents Handler

The `NotifyDBEvents` method is invoked by Oracle Objects for OLE when database events of interest are fired.

For more detailed information about setting up the queues and triggers for Oracle Database events, see to Triggers on System Events and User Events in *Oracle Database Concepts*.

The syntax of the method is:

```
Public Function NotifyDBEvents(ByVal Ctx As Variant, ByVal Payload As Variant
```

Variants

The variants for the method are:

Variants	Description
[in] *Ctx*	Passed into the OraSubscriptions.Add method by the application. Context-sensitive information that the application wants passed on to the dbevent handler.
[in] *Payload*	The payload for this notification. Database events are fired by setting up event trigger and queues. *Payload* here refers to the payload, if any, that was enqueued in the queue when the event triggered.

Examples

Example: Registering an Application for Notification of Database Events

In the following example, an application subscribes for notification of database logon events (such as all logons to the database). When a user logs on to the database, the NotifyDBEvents method of the DBEventsHdlr that was passed in at the time of subscription is invoked. The context-sensitive information and the event-specific information are passed into the NotifyDBEvents method.

The DBEventsHdlr in this example is DBEventCls, which is defined later.

The main application:

```
' First instantiate the dbevent handler. The dbevent notification
' will fire the NotifyDBEvents on the callback handler.

Public DBEventsHdlr As New DBEventCls
Private Sub Form_Load()
    Dim gOraSession As Object
    Dim gOraSubscriptions As OraSubscriptions
    Dim gOraDatabase As OraDatabase

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

   'Create the OraDatabase Object by opening a connection to Oracle.
    Set gOraDatabase = gOraSession.DbOpenDatabase
             ("ora90.us.oracle.com", "pubsub/pubsub",
              ORADB_ENLIST_FOR_CALLBACK)
    Set gOraSubscriptions = gOraDatabase.Subscriptions
    gOraSubscriptions.Add "PUBSUB.LOGON:ADMIN", DBEventsHdlr,
             gOraDatabase
    gOraSubscriptions(0).Register
    MsgBox "OK"
End Sub
```

The database event handler class that defines the NotifyDBEvents method.

```
Public countofMsgs as integer
Public Function NotifyDBEvents(Ctx As Variant, Payload As Variant )
    On error goto NotifyMeErr

    MsgBox "Retrieved payload " + Payload
   ' do something - here the subscription is unregistered after
   ' receiving 3 notifications
    countofMsgs = countofMsgs + 1
```

```
    If countofMsgs > 3 Then
        Ctx.Subscriptions(0).UnRegister
    End If
    Exit Sub
NotifyMeErr:
    Call RaiseError(MyUnhandledError, "newcallback:NotifyMe Method")

End Sub
```

See Also:

- "Database Events" on page 4-22 for a complete discussion of the concepts involved in this example
- Triggers on System Events and User Events in *Oracle Database Concepts* for detailed information about setting up the queues and triggers for Oracle Database Events
- OraSubscription Object on page 9-61
- Remove (OraSubscriptions Collection) Method on page 10-231

AddIntervalDS Method

Applies To

OraTimeStamp Object on page 9-62

OraTimeStampTZ Object on page 9-64

Description

Adds an interval that represents an interval from days to seconds, to the `OraTimeStamp` or `OraTimeStampTZ` object.

Usage

```
OraTimeStampObj.AddIntervalDS operand
OraTimeStampTZObj.AddIntervalDS operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object that represents an interval from days to seconds to be added to the current `OraTimeStamp` or `OraTimeStampTZ` object.

Remarks

The result of adding an interval to the current `OraTimeStamp` or `OraTimeStampTZ` object is stored in the current object, overwriting any previous value. There is no return value.

If *operand* is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If *operand* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` object represents.

Examples

Using OraTimeStamp

```
Dim OraTimeStamp As OraTimeStamp

...

'Create OraTimeStamp using a string
Set OraTimeStamp = OraSession.CreateOraTimeStamp("2000-12-28 00:00:00", _
         "YYYY-MM-DD HH:MI:SS")

'Add an interval using numeric value that represents 5 days and 12 hours
OraTimeStamp.AddIntervalDS 5.5

'Value should now be "2001-1-2 12:00:00"
tsStr = OraTimeStamp.Value
```

Using OraTimeStampTZ

```
Dim OraTimeStampTZ As OraTimeStampTZ

...

'Create OraTimeStampTZ using a string
Set OraTimeStamp = OraSession.CreateOraTimeStampTZ("2000-12-28 00:00:00 -07:00", _
       "YYYY-MM-DD HH:MI:SS TZH:TZM")

'Add an interval using numeric value that represents 5 days and 12 hours
OraTimeStampTZ.AddIntervalDS 5.5

'Value should now be "2001-1-2 12:00:00"
tstzStr = OraTimeStampTZ.Value

...
```

AddIntervalYM Method

Applies To

OraTimeStamp Object on page 9-62

OraTimeStampTZ Object on page 9-64

Description

Adds an interval that represents an interval from years to months, to the `OraTimeStamp` or `OraTimeStampTZ` object.

Usage

```
OraTimeStampObj.AddIntervalYM operand
OraTimeStampTZObj.AddIntervalYM operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object that represents an interval from years to months, to be added to the current `OraTimeStamp` or `OraTimeStampTZ` object.

Remarks

The result of adding an interval to the current `OraTimeStamp` or `OraTimeStampTZ` object is stored in the current object, overwriting any previous value. There is no return value.

If *operand* is a `Variant` of type `String`, it must be in following format: [+/-] YEARS-MONTHS.

If *operand* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

Examples

Example: Using the OraTimeStamp Object

```
Dim OraTimeStamp As OraTimeStamp

...
'Create OraTimeStamp using a string
Set OraTimeStamp = OraSession.CreateOraTimeStamp("2000-12-28 00:00:00", _
         "YYYY-MM-DD HH:MI:SS")

'Add an interval using numeric value that represents 2 years
OraTimeStamp.AddIntervalYM 2

'Value should now be "2002-12-28 00:00:00"
tsStr = OraTimeStamp.Value
```

```
...
```

Example: Using the OraTimeStampTZ Object

```
Dim OraTimeStampTZ As OraTimeStampTZ

...
'Create OraTimeStampTZ using a string
Set OraTimeStampTZ =OraSession.CreateOraTimeStampTZ("2000-12-28 00:00:00" & _
            "-07:00"  "YYYY-MM-DD HH:MI:SS TZH:TZM")

'Add an interval using numeric value that represents 2 years
OraTimeStampTZ.AddIntervalYM 2

'Value should now be "2002-12-28 00:00:00"
tstzStr = OraTimeStampTZ.Value

...
```

AddNew Method

Applies To

OraDynaset Object on page 9-30

Description

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

Usage

```
oradynaset.AddNew
oradynaset.DbAddNew
```

Remarks

When an `AddNew` operation is initiated, values of fields present within the dynaset are maintained in a copy buffer and do not reflect the actual contents of the database.

The values of the fields are modified through the `OraField` object, and committed with an `Update` operation 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 the `AddNew` method using the `"INSERT into TABLE (...) VALUES (...)"` SQL statement, and are added to the end of the table.

When adding a row that has object, collection, and `REF` columns, these column values should be set to a valid `OraObject`, `OraCollection`, or `OraRef` interface or to the `Null` value. The column values can also be set with the automation object returned by the `CreateOraObject` method. When adding a row having a `BLOB`, `CLOB`, or `BFILE` column, the column value should be set to a valid `OraBLOB`, `OraCLOB`, or `OraBFILE` interface, `Null`, or `Empty`. Setting a `BLOB`, `CLOB`, and `BFILE` column to an `Empty` value inserts an empty LOB value into the database.

> **Note:** A call to `Edit`, `AddNew`, or `Delete` methods cancels any outstanding `Edit` or `AddNew` method calls before proceeding. Any outstanding changes not saved using an `Update` method are lost during the cancellation.

Examples

This example demonstrates the use of the `AddNew` and `Update` methods 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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
```

```
Dim OraDynaset As OraDynaset

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

'Create the OraDatabase Object by opening a connection to Oracle.

Set OraDatabase = OraSession.OpenDatabase("ExampleDb", _
              "scott/tiger", 0&)

'Create the OraDynaset Object.
Set OraDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)
'Begin an AddNew.
OraDynaset.AddNew

'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.Update

MsgBox "Added one new employee."

End Sub
```

See Also:

- Delete Method on page 10-116
- Edit Method on page 10-134
- EditMode Property on page 11-52
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- Update Method on page 10-257
- Validate Event on page 12-9

AddTable Method

Applies To

OraParameters Collection on page 9-68

Description

Adds an array parameter to the `OraParameters` collection.

Usage

```
oraparamarray.AddTable  Name, IOType, ServerType, ArraySize , ElementSize,
ObjectName
```

Arguments

The arguments for the method are:

Arguments	Description
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.
IOType	An integer code specifying how the parameter is to be used in SQL statements and PL/SQL blocks.
ServerType	Specifies Oracle Database type to which this array parameter is to be bound. For a list of possible values, see the `OraParameter` ServerType Property on page 11-138.
ArraySize	Defines the number of elements in the parameter array. This parameter is used to calculate the maximum buffer length.
ElementSize [optional]	Defines the size of the element. Valid for only character and string type table (array) parameters. The valid size for *ElementSize* depends on the `VarType`. *ElementSize* is optional in all cases except when bound to `char` and `string` types.
ObjectName	A case-sensitive string containing the name of the `Object`. This is only required if *ServerType* is `ORATYPE_OBJECT`, `ORATYPE_VARRAY`, or `ORATYPE_TABLE`. It is required for `ORATYPE_REF` when the `REF` is used in PL/SQL.

IO Type Settings

The *IOType* settings are:

Constant	Value	Description
`ORAPARM_INPUT`	1	Used for input variables only.
`ORAPARM_OUTPUT`	2	Used for output variables only.
`ORAPARM_BOTH`	3	Used for variables that are both input and output.

Verify that this value is correct. If you set an incorrect option, such as `ORAPARM_BOTH` for the stored procedure parameter type `IN`, this can result in errors. `ORAPARM_BOTH` is for `IN` and `OUT` parameters only. It is not used against one stored procedure that has

an IN parameter and another that has an OUT parameter. In this case, use two parameters. Errors caused in this way are rare, but if there are parameter-related errors, verify that the *IOType* is correct.

Server Type

See ServerType Property on page 11-138 for valid types and note the following:

Note:

- External data type ORATYPE_NUMBER allows decimal precision of 1 to 38.
- The maximum positive number is 0.99999999999999999999 E + 38.
- The minimum positive number is 0.1 E-38.
- The minimum negative number is -0.99999999999999999999 E + 38.
- The maximum negative number is 0.1 E -38.

ElementSize (Optional)

Valid for character, string, and raw types. The valid size for *ElementSize* depends on the *VarType*. This represents the length of each individual string or raw array element. These ranges are listed.

VarType	Size
ORATYPE_VARCHAR2	Valid range from 1 to 1999
ORATYPE_VARCHAR	Valid range from 1 to 1999
ORATYPE_STRING	Valid range from 1 to 1999
ORATYPE_CHAR	Valid range from 1 to 255
ORATYPE_CHARZ	Valid range from 1 to 255
ORATYPE_RAW_BIN	Valid range from 1 to 4000 (see remarks)

Remarks

Use parameters to represent SQL bind variables for array insert, update, and delete operations, rather than rebuilding the SQL statement. SQL bind variables are useful because you can change a parameter value without having to parse the query again. 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.

The ServerType value ORATYPE_RAW_BIN is used when binding to Oracle Raw columns. A byte array is used to Put or Get values. The maximum allowable size of ORATYPE_RAW_BIN bind buffers is 2000 bytes when bound to a column of a table: the maximum allowable size is 32 KB when bound to a stored procedure. No element (see *ElementSize* argument) can be greater than 4000 bytes when binding to stored procedures, 2000 bytes against columns of tables. For example code, see the samples in the *ORACLE_BASE**ORACLE_HOME*\OO4O\VB\Raw directory.

Examples

See "Example: Using OraParamArrays with PL/SQL" on page 9-49.

See Also: ServerType Property on page 11-138

Append (OraCollection) Method

Applies To

OraCollection Object on page 9-19

Description

Extends the size of the collection by one and appends the `Variant` value at the end of the collection.

Usage

```
OraCollection.Append element
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *element*	A `Variant` representing the value to be appended.

Remarks

If an `OraCollection` represents a collection of `Object` types or `Refs`, the element argument should represent a valid `OraObject` or `OraRef`.

Examples

The following example illustrates the `Append` method. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3.

Example: Append Method for the OraCollection Object Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameList as OraCollection

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

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

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'retrieve a Enames column from Department.
'Here Value property of OraField object returns EnameList OraCollection
set EnameList = OraDynaset.Fields("Enames").Value

'Append an "Eric" to the collection.
'Before that row level lock should be obtained
OraDynaset.Edit
EnameList.Append "Eric"
```

```
OraDynaset.Update
```

Append (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Appends the LOB content of the input OraLOB object to the internal LOB value of this instance.

Usage

```
OraBlob.Append srcBlob
OraClob.Append srcClob
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *srcLOB*	A valid object of type OraBLOB or OraCLOB.

Remarks

Appends the LOB content of input LOB to the end of current LOB value. Obtain either a row-level lock or an object-level lock before calling this method.

AppendChunk Method

Applies To

OraField Object on page 9-33

Description

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

Usage

```
orafield.AppendChunk(string)
orafield.DbAppendChunk(string)
```

Arguments

The arguments for the method are:

Arguments	Description
string	Data to append to the specified field.

Remarks

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

Examples

Note: This example cannot be run as is. It requires a defined form named frmChunk.

This example demonstrates the use of the AppendChunk method 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 named frmChunk. 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.AddNew

 '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").AppendChunk (CurChunk)

 Next I

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

 'Close the file.
 Close FNum

 frmChunk.MousePointer = DEFAULT

End Sub
```

See Also:

- "Migration from LONG RAW to LOB or BFILE" on page 5-5
- FieldSize Method on page 10-150
- GetChunk Method on page 10-156
- Type Property on page 11-164

AppendChunkByte Method

Applies To

OraField Object on page 9-33

Description

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

Usage

```
orafield.AppendChunkByte(ByteArray, numbytes)
```

Arguments

The arguments for the method are:

Arguments	Description
Byte Array	Data to append to the specified field.
numbytes	Number of bytes to copy.

Remarks

The AppendChunkByte method allows the manipulation of data fields that are larger than 64 KB.

Examples

> **Note:** This is an incomplete code sample, provided for your reference. A complete Visual Basic sample called LONGRAW that is based on this code sample, is provided in the OO4O samples directory.

This sample code demonstrates the use of the AppendChunkByte method to read a file into a LONG RAW column of a database. This code expects a valid dynaset named OraDynaset representing a table with a column named longraw.

```
Sub AppendChunkByteExample (FName As String)

 'Declare various variables.
 Dim NumChunks As Integer, RemChunkSize As Integer
 Dim TotalSize As Long, CurChunkByte() As Byte
 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.AddNew
 '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

  ReDim CurChunkByte(ChunkSize)

  'Read a chunk from the file.
  Get #FNum, , CurChunkByte

  'Append chunk to LONGRAW field.
  OraDynaset.Fields("LONGRAW").AppendChunkByte (CurChunkByte)
 Next I

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

 'Close the file.
 Close FNum

 frmChunk.MousePointer = DEFAULT

End Sub
```

See Also: "Migration from LONG RAW to LOB or BFILE" on page 5-5

AQAgent (OraAQMsg) Method

Applies To

OraAQMsg Object on page 9-6

Description

Creates an instance of the `OraAQAgent` for the specified consumer and adds it to the `OraAQAgents` list of the message.

Usage

```
Set agent = qMsg.AQAgent(name)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *name*	A `String` up to 30 bytes representing the name of the consumer of the message.
[in] [optional] *Address*	A 128-byte `String` representing the protocol specific address of a recipient, such as [`schema`.]`queue`[`@dblink`].

Remarks

The `OraAQAgent` object represents a message recipient and is only valid for queues that allow multiple consumers. Queue subscribers are recipients by default. Use this object to override the default consumers.

An `OraAQAgent` object can be instantiated by invoking the `AQAgent` method. For example:

```
Set agent = qMsg.AQAgent(consumer)
```

The maximum number of agents that a message can support is 10.

The `AQAgent` method returns an instance of an `OraAQAgent` object.

> **Note:** *Address* is not supported in this release, but is provided for future enhancements.

AQMsg (OraAQ) Method

Applies To

OraAQ Object on page 9-3

Description

Creates an `OraAQMsg` for the specified options.

Usage

```
Set qMsg = Q.AQMsg(msgtype, typename, schema)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *msgtype*	An `Integer` representing a `RAW` or user-defined type. Optional for `RAW` type. Possible values are: ■ `ORATYPE_RAW` (`23`) - Message type is `RAW`. ■ `ORATYPE_OBJECT` (`108`) - Message type is user-defined.
[in] *typename*	A `String` representing the name of the type. Optional for `RAW` type. Default is `'RAW'`.
[in] [optional] *schema*	A `String` representing the schema where the type is defined. Default is `'SYS'`.

Remarks

The method could be used as follows:

```
set QMsg = Q.AQMsg(ORATYPE_OBJECT,"MESSAGE_TYPE","SCOTT")
set QMsg = Q.AQMsg
```

ArcCos (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the arc cosine of an `OraNumber` object. The result is in radians.

Usage

```
OraNumber.ArcCos
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method returns an error if the `OraNumber` value is less than `-1` or greater than `1`.

ArcSin (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the arc sine of an `OraNumber` object. Result is in radians.

Usage

```
OraNumber.ArcSin
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method returns an error if the `OraNumber` object is less than `-1` or greater than `1`.

ArcTan (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the arc tangent of an `OraNumber` object. Result is in radians.

Usage

```
OraNumber.ArcTan
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

ArcTan2 (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the arc tangent of two numbers using the *operand* provided. The result is in radians.

Usage

```
OraNumber.ArcTan2 operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method returns an error if *operand* is zero.

Attribute (OraMetaData) Method

Applies To

OraMetaData Object on page 9-39

Description

Returns the `OraMDAttribute` object at the specified *`index`*.

Usage

```
Set OraMDAttribute = OraMetaData.Attribute(2)
Set OraMDAttribute = OraMetaData.Attribute("AttributeName")
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *`index`*	An `Integer` index between `0` and `count-1`, or a `String` representing the name of an attribute.

Remarks

None.

See Also: OraMetaData Object on page 9-39 for a list of possible attribute names

AutoBindDisable Method

Applies To

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

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.

Examples

This example demonstrates the use of the `AutoBindDisable` and `AutoBindEnable` methods 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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. OpenDatabase("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.CreateDynaset("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.CreateDynaset("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
```

See Also: AutoBindEnable Method on page 10-41

AutoBindEnable Method

Applies To

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

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.

Examples

This example demonstrates the use of the `AutoBindDisable` and `AutoBindEnable` methods 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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession. OpenDatabase("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.CreateDynaset("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.CreateDynaset("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
```

See Also: AutoBindDisable Method on page 10-39

BeginTrans Method

Applies To

OraConnection Object on page 9-27

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

Begins a database transaction within the specified session.

Usage

```
oraconnection.BeginTrans
oradatabase.BeginTrans
orasession.BeginTrans
```

Remarks

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

If `Update` or `Delete` methods fail on a given row in a dynaset in a global transaction after you issue a `BeginTrans`, be aware that locks remain on those rows on which you called the `Update` or `Delete` method. These locks persist until you call a `CommitTrans` or `Rollback` method.

> **Note:** If an `OraDatabase` object has been enlisted with Microsoft Transaction Server (MTS) and is part of a global MTS transaction, this method has no effect.

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset
 Dim fld As OraField

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

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

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

 'Start Transaction processing.
 OraSession.BeginTrans

 '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.Edit
   fld.value = 0
   OraDynaset.Update
   OraDynaset.MoveNext
 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.Rollback
 'MsgBox "Salary changes rolled back."

End Sub
```

See Also:

- AutoCommit Property on page 11-9
- CommitTrans Method on page 10-66
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235
- "Microsoft Transaction Server Support" on page 3-15

Cancel Method

Applies To

OraSQLStmt Object on page 9-60 created with the ORASQL_NONBLK option

Description

Cancels the currently executing SQL operation.

Usage

```
status = OraSQL.NonBlockingState
   if status = ORASQL_STILL_EXECUTING
OraSQL.Cancel
Endif
```

Return Values

ORASQL_SUCCESS(0) - Any errors are thrown as exceptions.

See Also: "Asynchronous Processing" on page 3-16

CancelEdit (OraRef) Method

Applies To

OraRef Object on page 9-52

Description

Unlocks the referenceable object in the database and cancels the object update operation.

Usage

```
OraRef.CancelEdit
```

Remarks

Care should be taken before using this method; it cancels any pending transaction on the connection.

Ceil (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the ceiling value of an `OraNumber` object.

Usage

```
OraNumber.Ceil
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

ChangePassword (OraServer) Method

Applies To

OraServer Object on page 9-56

Description

Changes the password for a given user.

Usage

```
OraServer.ChangePassword user_name, current_password, new_password
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *user_name*	A `String` representing the user for whom the password is changed.
[in] *current_password*	A `String` representing the current password for the user.
[in] *new_password*	A `String` representing the new password for whom the user account is set.

Remarks

The `OraServer` object should be attached to an Oracle database using the `Open` method before to using this method.

This method is useful when a password has expired. In that case, the `OpenDatabase` method could return the following error:

```
ORA-28001 "the password has expired".
```

See Also:

- BeginTrans Method on page 10-43
- Close Method on page 10-63
- CommitTrans Method on page 10-66
- CreateAQ Method on page 10-79
- CreateCustomDynaset Method on page 10-80
- CreateTempBLOB/CLOB Method on page 10-114
- CreateDynaset Method on page 10-85
- CreateOraObject (OraDatabase) Method on page 10-97
- CreateSQL Method on page 10-111
- Describe Method on page 10-124
- ExecuteSQL Method on page 10-144
- FetchOraRef Method on page 10-149
- LastServerErrReset Method on page 10-189
- MonitorForFailover Method on page 10-194
- Open (OraServer) Method on page 10-210
- OpenDatabase Method on page 10-212
- RemoveFromPool Method on page 10-232
- Rollback Method on page 10-235

ChangePassword (OraSession) Method

Applies To

OraSession Object on page 9-58

Description

Changes the password for a given user.

Usage

```
OraSession.ChangePassword database_name, user_name, current_password, new_password
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *database_name*	A `String` representing the Oracle network specifier used when connecting to a database.
[in] *user_name*	A `String` representing the user for whom the password is changed.
[in] *current_password*	A `String` representing the current password for the user.
[in] *new_password*	A `String` representing the new password for whom the user account is set.

Remarks

This method is especially useful when a password has expired. In that case, the `OpenDatabase` or `CreateDatabasePool` method could return the following error:

```
ORA-28001 "the password has expired".
```

Examples

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim password as String

'Note: The DBA could expire scott's password by issuing
'ALTER USER SCOTT PASSWORD EXPIRE

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
password = "tiger"

On Error GoTo err:
Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/" & password, 0&)
End

err:
'Check for password expiration error

If OraSession.LastServerErr = 28001 Then
    OraSession.ChangePassword "ExampleDb", "scott", password, "newpass"
    'reset our password variable, then try OpenDatabase again
```

```
    password = "newpass"
    Resume
End If

End
```

See Also:

- OpenDatabase Method on page 10-212
- CreateDatabasePool Method on page 10-83

Clone Method

Applies To

OraDynaset Object on page 9-30

Description

Returns a duplicate dynaset of the specified dynaset.

Usage

```
Set oradynaset2 = oradynaset1.Clone
Set oradynaset2 = oradynaset1.DbClone
```

Remarks

This method creates a duplicate dynaset of the one specified. The original and duplicate dynasets have their own current record. However, the new dynaset is not positioned on any row and has its `EOF` and `BOF` conditions set to `True`. To change this, you must explicitly set a current row on the new duplicate with a `Move` or `Find` method.

Using the `Clone` method has no effect on the original dynaset. You cannot add, update, or remove records from a dynaset clone.

Use the `Clone` method to perform an operation on a dynaset that requires multiple current records.

A cloned dynaset does not have all the property settings of the original. The `CacheBlock`, `CacheSliceSize`, `CacheSlicePerBlock`, and `FetchLimit` properties are all set to `Null`.

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

See Also:

- Bookmark Property on page 11-13
- CreateDynaset Method on page 10-85

Clone (OraLOB/BFILE) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Returns the clone of an OraLOB or `OraBFILE` object.

Usage

```
OraBlob1 = OraBlob.Clone
OraClob1 = OraClob.Clone
OraBfile = OraBfile.Clone
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *OraLOB*	A valid object of type `OraBLOB`, `OraCLOB`, or `OraBFILE`.

Remarks

This method makes a copy of an `OraBLOB` or `OraCLOB` object. This copy does not change due to a dynaset move operation or `OraSQLStmt Refresh` operation. No operation that modifies the LOB content of an `OraBLOB` or `OraCLOB` object can be performed on a clone.

This method makes a copy of Oracle `BFILE` locator and returns an `OraBFILE` associated with that copy. The copy of an `OraBFILE` does not change due to a dynaset move operation or a `OraSQLStmt` refresh operation.

Clone (OraCollection) Method

Applies To

OraCollection Object on page 9-19

Description

Returns the clone of an `OraCollection` object.

Usage

```
set OraCollection1 = OraCollection.Clone
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *oraCollection1*	A valid `OraCollection` object

Remarks

This method makes a copy of an Oracle collection and returns an `OraCollection` object associated with that copy. This copy of an Oracle collection does not change due to a dynaset move operation or `OraSQLStmt Refresh` operation. An `OraCollection` object returned by this method allows operations to access its element values of the underlying Oracle collection and prohibits any operation that modifies its element values.

Clone (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Returns a copy of the `OraIntervalDS` object.

Usage

```
Set OraIntervalDSObjClone = OraIntervalDSObj.Clone
```

Remarks

Returns a new `OraIntervalDS` object with the same value as the original.

Clone (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Returns a copy of the `OraIntervalYM` object.

Usage

```
Set OraIntervalYMObjClone = OraIntervalYMObj.Clone
```

Remarks

Returns a new `OraIntervalYM` object with the same value as the original.

Clone (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Returns a copy of the `OraNumber` object .

Usage

```
Set OraNumber2 = OraNumber.Clone
```

Remarks

Returns a new `OraNumber` object with the same value as the original.

Clone (OraObject/Ref) Method

Applies To

OraObject Object on page 9-43

OraRef Object on page 9-52

Description

Returns the clone of an `OraObject` or `OraRef` object.

Usage

```
Set OraObjectClone = OraObject.Clone
Set OraRefClone = OraRef.Clone
```

Remarks

This method makes a copy of a `Value` instance or `REF` value and returns an `OraObject` or `OraRef` object associated with that copy. This copy does not change due to a dynaset move operation or `OraSQLStmt` refresh operation. An `OraObject` object returned by this method allows an operation to access its attribute values of an underlying value instance and disallows any operation to modify its attribute values.

Examples

Before running the sample code, make sure that you have the necessary data types and tables in the database. For the following examples, see "Schema Objects Used in the OraObject and OraRef Examples" on page A-3

Example: Clone Method for the OraObject Object

The following example shows the use of the `Clone` method.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Address as OraObject
Dim AddressClone as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab",0&)

'retrieve a address column from person_tab. Here Value property of OraField object
'returns Address OraObject
set Address = OraDynaset.Fields("Addr").Value

'here Address OraObject points to Address value instance in the server
'for the first row
msgbox Address.Street
```

```
'move to second row
OraDynaset.MoveNext

'here Address OraObject points to Address value instance in the server
'for the second row
msgbox Address.Street

'get the clone of Address object. This clone points to the copy of
'the value instance for second row
set AddressClone = Address.Clone

'move to third row
OraDynaset.MoveNext

'here Address OraObject points to Address value instance in the server
'for third row
msgbox Address.Street

'here AddressClone OraObject points to copy of Address value instance
' in the server for second row
msgbox AddressClone.Street
```

Example: Clone Method for the OraRef Object

The following example shows the usage of the `Clone` method. Before running the sample code, make sure that you have the necessary data types and tables in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef
Dim PersonClone as OraRef

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)

'retrieve a aperson column from customers.
'Here Value property of OraField object 'returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'here Person OraRef points to Person Ref value in the server for the first row
msgbox Person.Name

'move to second row
OraDynaset.MoveNext

'here Person OraRef points to Person Ref value in the server for the second row
msgbox Person.Name

'get the clone of Person object.
'This clone points to the copy of the Ref for second row
set PersonClone = Person.Clone
```

```
'move to third row
OraDynaset.MoveNext

'here Person OraRef points to Person Ref value
'in the server for the third row
msgbox Person.Name

'here PersonClone OraRef points to Person Ref value
'in the server for the second row
msgbox PersonClone.Name
```

Clone (OraTimeStamp) Method

Applies To

OraTimeStamp Object on page 9-62

Description

Returns a copy of the `OraTimeStamp` object.

Usage

```
Set OraTimeStampObj1 = OraTimeStampObj.Clone
```

Remarks

Returns a new `OraTimeStamp` object with the same value as the current object.

Clone (OraTimeStampTZ) Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns a copy of the `OraTimeStampTZ` object.

Usage

```
Set OraTimeStampTZObj1 = OraTimeStampTZObj.Clone
```

Remarks

Returns a new `OraTimeStampTZ` object with the same value as the current object.

Close Method

Applies To

OraDatabase Object on page 9-28

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

OraServer Object on page 9-56

Description

Does nothing. Added for compatibility with Visual Basic.

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, the object closes automatically.

See Also:

- CreateDynaset Method on page 10-85
- OpenDatabase Method on page 10-212

Close (OraBFILE) Method

Applies To

OraBFILE Object on page 9-9

Description

Closes an opened `BFILE` data type.

Usage

```
OraBfile = OraBfile.Close
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *OraBfile*	A valid object of type `OraBFILE`.

Remarks

This method only applies to `BFILE`s, not LOBs.

CloseAll (OraBFILE) Method

Applies To

OraBFILE Object on page 9-9

Description

This method closes all open `OraBFILE` objects on this connection.

Usage

```
OraBfile.CloseAll
```

CommitTrans Method

Applies To

OraConnection Object on page 9-27

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

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

Usage

```
oraconnection.CommitTrans
oradatabase.CommitTrans
orasession.CommitTrans
```

Remarks

The `CommitTrans` method acts differently for these objects:

- `OraConnection` and `OraDatabase`

  The `CommitTrans` method commits all pending transactions for the specified connection. This method has no effect if a transaction has not started. When a sessionwide transaction is in progress, you can use this method to commit the transactions for the specified connection prematurely.

- `OraSession`

  The `CommitTrans` method commits all transactions present within the session. The `CommitTrans` method is valid only when a transaction has been started. If a transaction has not been started, using the `CommitTrans` method causes an error.

Note: If an `OraDatabase` object has been enlisted with Microsoft Transaction Server (MTS) and is part of a global MTS transaction, this method has no effect.

Examples

This example demonstrates the use of the `BeginTrans` method to group a set of dynaset edits into a single transaction. `The CommitTrans` method then accepts the changes. Copy this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 Dim fld As OraField

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

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

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

 'Start Transaction processing.
 OraSession.BeginTrans

 '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.Edit
   fld.value = 0
   OraDynaset.Update
   OraDynaset.MoveNext
 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.CommitTrans
 MsgBox "Salary changes committed."

End Sub
```

See Also:

- AutoCommit Property on page 11-9
- BeginTrans Method on page 10-43
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235
- "Microsoft Transaction Server Support" on page 3-15

Compare (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Compares the specified portion of the LOB value of an `OraBLOB` or `OraCLOB` object (or `OraBFILE` object) to the LOB value of the input `OraBLOB` or `OraCLOB` object (or `OraBFILE` object).

Usage

```
IsEqual = OraBlob.Compare srcBlob, amount, Offset, srcOffset
IsEqual = OraClob.Compare srcClob, amount, Offset, srcOffset
IsEqual = OraBfile.Compare srcBfile, amount, Offset, srcOffset
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *srcLOB*	Input `OraBLOB`, `OraCLOB`, or `OraBFILE` object whose value is to be compared.
[in] [optional] *amount*	An `Integer` specifying the number of bytes or characters to compare. The default value of *amount* is from the *Offset* to the end of each LOB.
[in] [optional] *Offset*	An `Integer` specifying the `1`-based *Offset* in bytes (`OraBLOB` or `OraBFILE`) or characters (`OraCLOB`) in the value of this object. Default value is `1`.
[in] [optional] *srcOffset*	An `Integer` specifying the `1`-based *Offset* in bytes (`OraBLOB` or `OraBFILE`) or characters (`OraCLOB`) in the value of the `srcLob` object. Default value is `1`.
[out] *IsEqual*	A Boolean representing the result of a compare operation.

Remarks

The `Compare` method returns `True` if comparison succeeds; otherwise, it returns `False`.

If the amount to be compared causes the comparison to take place beyond the end of one LOB but not beyond the end of the other, the comparison fails. Such a comparison could succeed only if the amount of data from the *Offset* to the end is the exactly the same for both LOBs.

This call is currently implemented by executing a PL/SQL block that utilizes `DBMS_LOB.INSTR()`.

ConnectSession Method

Applies To

OraSession Object on page 9-58

Description

Returns the `OraSession` object with the specified name that is associated with the `OraClient` object of the specified session.

Usage

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

Arguments

The arguments for the method are:

Arguments	Description
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 `OraClient` object of the current session and searching for a session named `session_name`. The `OraSessions` collection contains only sessions created through the current application. This means that it is not possible to share sessions across applications, only within applications.

Examples

This example demonstrates the use of the `ConnectSession` and `CreateNamedSession` methods 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
 Dim dfltsess As OraSession
 Dim OraSession As OraSession

 '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
```

See Also:

- CreateSession Method on page 10-109
- OraClient Object on page 9-18
- OraSessions Collection on page 9-69

CopyToClipboard Method

Applies To

OraDynaset Object on page 9-30

Description

Copy the rows from the dynaset to the clipboard in text format.

Usage

```
OraDynaset.CopyToClipboard(NumOfRows, colsep, rowsep)
```

Arguments

The arguments for the method are:

Arguments	Description
NumOfRows	Number of rows to be copied to the dynaset
colsep [optional]	Column separator in the CHAR data type to be inserted between columns
rowsep [optional]	Row separator in the CHAR data type to be inserted between rows

Remarks

This method is used to help transfer data between the Oracle Object for OLE cache (dynaset) and Windows applications, such as Excel or Word. The CopyToClipboard method copies data starting from the current position of the dynaset up to the last row.

The default column separator is TAB (ASCII 9).

The default row separator is ENTER (ASCII 13).

Examples

The following example copies data from the dynaset to the clipboard. Paste this code into the definition section of a form, then press **F5**.

```
Sub Form_Load ()
 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

'Now call CopyToClipboard to copy the entire dynaset
 OraDynaset.CopyToClipboard -1, chr(9), chr(13)
End Sub
```

Copy (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Copies a portion of the internal LOB value of an input `OraBLOB` or `OraCLOB` object to internal LOB value of this instance.

Usage

```
OraBlob.Copy srcBlob, amount, destOffset, srcOffset
OraClob.Copy srcClob, amount, destOffset, srcOffset
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *srcLOB*	An `OraCLob` or `OraBLOB` object whose value is to be copied.
[in] [optional] *amount*	An `Integer` specifying number of bytes or characters to copy. Default value is the size of the `BLOB` or `CLOB` value of the *srcLOB* object.
[in] [optional] *destOffset*	An `Integer` specifying the offset in bytes or characters for the value of this object. Default value is `1`.
[in] [optional] *srcOffset*	An `Integer` specifying the offset in bytes or characters, for the value of the *srcLOB* object. Default value is `1`.

Remarks

Obtain either a row-level lock or object-level lock before calling this method.

CopyFromFile (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Loads or copies a portion or all of a local file to the internal LOB value of this object.

Usage

```
OraBlob.CopyFromFile "blob.bmp" amount, offset, chunksize
OraClob.CopyFromFile "clob.txt" amount, offset, chunksize
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *filename*	A string specifying the absolute name and path for the file to be read.
[in] [optional] *amount*	An Integer specifying the maximum number in bytes to be copied. Default value is total file size.
[in] [optional] *offset*	An Integer specifying the absolute offset of the BLOB or CLOB value of this object, in bytes for OraBLOB or OraBFILE and characters for OraCLOB. Default value is 1.
[in] [optional] *chunksize*	An Integer specifying the size for each read operation, in bytes. If chunksize parameter is not set or 0, the value of the *amount* argument is used, which means the entire amount is transferred in one chunk.

Remarks

Obtain either a row-level lock or object-level lock before calling this method.

The file should be in the same format as the NLS_LANG setting.

> **Note:** When manipulating LOBs using LOB methods, such as Write and CopyFromFile, the LOB object is not automatically trimmed if the length of the new data is smaller than the old data. Use the Trim (OraLOB) method to shrink the LOB object to the size of the new data.

Examples

Example: Using the CopyFromFile Method

This example demonstrates the use of the CopyFromFile method.

Be sure that you have the PART table in the database with valid LOB data in it. Also, be sure that you have installed the OraLOB Schema Objects as described in "Schema Objects Used in LOB Data Type Examples" on page A-3.

```
Dim OraSession As OraSession
```

```
Dim OraDatabase As OraDatabase
Dim PartImage as OraBLOB

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

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

'Create a Dynaset containing a BLOB and a CLOB column
set part = OraDatabase.CreateDynaset ("select * from part where" & _
               "part_id = 1234",0)
set PartImage = part.Fields("part_image").Value

'copy the entire content of partimage.jpg file to LOBS
part.Edit
PartImage.CopyFromFile "partimage.jpg"
part.Update
```

See Also: Trim (OraLOB) Method on page 10-254

CopyFromBFILE (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Copies a portion or all of the LOB value of an `OraBFILE` object to the LOB value of this object.

Usage

```
OraBlob.CopyFromBFile srcBFile, amount, destOffset, srcOffset

OraClob.CopyFromBFile srcBFile, amount, destOffset, srcOffset
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *srcBFile*	An `OraBFILE` object from which the data is to be copied.
[in] [optional] *amount*	An `Integer` specifying the maximum number to be copied, in characters for `OraCLOB` or bytes for `OraBLOB` or `OraBFILE`. Default value is the size of `BFILE` value of the `srcBFile` object.
[in] [optional] *destOffset*	An `Integer` specifying the absolute offset for this instance. Default is `1`.
[in] [optional] *srcOffset*	An `Integer` specifying the absolute offset for the `BFILE` value of the source `OraBFILE` object. Default is `1`.

Remarks

Obtain either a row-level lock or object-level lock before calling this method.

For a single-byte character set, the `OraBFile` object should be of the same character set as the database.

If the database has a variable width character set, the `OraBFile` object passed to the `OraClob.CopyFromBFile` method must point to a file that uses the UCS2 character set.

CopyToFile (OraLOB/BFILE) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Copies a portion or all of the internal LOB value of this object to the local file.

Usage

```
OraBlob.CopyToFile "blob.bmp" amount,offset,chunksize
OraClob.CopyToFile "clob.txt" amount,offset,chunksize
OraBfile.CopyToFile "bfile.bmp" amount,offset,chunksize
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *filename*	A String specifying the absolute name and path for which the file is to be written.
[in] [optional] *amount*	An Integer specifying the maximum amount to be copied, in bytes for OraBLOB/OraBFILE and characters for OraCLOB. Default value is the size of the LOB or BFILE.
[in] [optional] *offset*	An Integer specifying absolute offset of the LOB or BFILE value of this instance, in bytes for OraBLOB/OraBFILE and characters for OraCLOB. Default value is 1.
[in] [optional] *chunksize*	An Integer specifying the size, in bytes, for each write operation. If the *chunksize* parameter is not set or is 0, the value of the *amount* argument is used which means the entire amount is transferred in one chunk.

Remarks

The file is in the same format as the NLS_LANG setting.

If the file exists, its contents is overwritten.

Examples

Example:Using the CopyToFile Method

This example demonstrates the use of the CopyToFile method.

Be sure that you have the PART table in the database with valid LOB data in it. Also, be sure that you have installed the OraLOB Schema Objects as described in "Schema Objects Used in LOB Data Type Examples" on page A-3.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim PartDesc as OraCLOB

'Create the OraSession Object.
```

```
Set OraSession = CreateObject("OracleInProcServer.XOraSession")

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

'Create a Dynaset containing a BLOB and a CLOB column
set part = OraDatabase.CreateDynaset ("select * from part where" & _
               "part_id = 1234",0)
set PartDesc = part.Fields("part_desc").Value

'Copy the entire LOB content to partdesc.txt file
PartDesc.CopyToFile "partdesc.txt"
```

Cos (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the cosine of an `OraNumber` object given in radians.

Usage

```
OraNumber.Cos
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

CreateAQ Method

Applies To

OraDatabase Object on page 9-28

Description

Creates an instance of the `OraAQ` object.

Usage

```
Set OraAq = OraDatabase.CreateAQ(Qname)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *Qname*	A `String` representing the name of the queue in the database.

Remarks

None.

CreateCustomDynaset Method

Applies To

OraDatabase Object on page 9-28

Description

Creates a dynaset using custom cache and fetch parameters

Usage

```
Set oradynaset = oradatabase.CreateCustomDynaset(sql_statement, options,
slicesize, perblock, blocks, FetchLimit, FetchSize, SnapShotID)
```

Arguments

The arguments for the method are:

Arguments	Description
sql_statement	Any valid Oracle SQL `SELECT` statement.
slicesize	Cache slice size.
perblock	Cache slices for each block.
blocks	Cache maximum number of blocks.
FetchLimit	Fetch array size.
FetchSize	Fetch array buffer size.
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. Specifying the constant `ORADYN_DEFAULT` or the value `&H0&` gives the following defaults for the dynaset: ■ Behave like Visual Basic Mode for a database: Field values not explicitly set are set to `Null`, overriding database column defaults. ■ Perform automatic binding of database parameters. ■ Remove trailing blanks from character string data retrieved from the database. ■ Create an updatable dynaset. ■ Cache data on the client. ■ Force a `MoveFirst` operation when the dynaset is created. ■ Maintain read-consistency.
SnapShotID [optional]	The ID of a Snapshot obtained from the `SnapShot` property of an `OraDynaset`.

Constants

The following table lists constants and values for the options flag.

Constant	Value	Description
`ORADYN_DEFAULT`	`&H0&`	Accept the default behavior.

Constant	Value	Description
ORADYN_NO_AUTOBIND	&H1&	Do not perform automatic binding of database parameters.
ORADYN_NO_BLANKSTRIP	&H2&	Do not remove 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.
ORADYN_ORAMODE	&H10&	Same as Oracle Mode for a database except it affects only the dynaset being created. If database was created in Oracle Mode, the dynaset inherits the property from it (for compatibility).
ORADYN_NO_REFETCH	&H20&	Behaves same as ORADB_NO_REFETCH mode for a database except this mode affects only the dynaset being created. If the database was created in ORADB_NO_REFETCH mode, the dynaset inherits the property for compatibility.
ORADYN_N_MOVEFIRST	&H40&	Does not force a MoveFirst when the dynaset is created. BOF and EOF are both true.
ORADYN_DIRTY_WRITE	&H80&	Update and Delete methods do not check for read consistency.

These values can be found in the oraconst.txt file located in:

*ORACLE_BASE**ORACLE_HOME*\rdbms\oo4o

Remarks

The SQL statement must be a SELECT statement or an error is returned. Features such as simple views and synonyms can be used freely. You can also use schema references, column aliases, table joins, nested select statements, and remote database references, but in each case you end up with a read-only dynaset.

If you use a complex expression or SQL function on a column, such as "sal + 100" or "abs(sal)", you get an updatable dynaset, but the column associated with the complex expression is not updatable.

Object names generally are not modifed, but in certain cases, they can be changed. For example, if you use a column alias, you must use the alias to refer to the field by name. If you use spaces in a complex expression, you must refer to the column without the spaces, because the database removes spaces. Note that you can always refer to a field by number, that is, by its ordinal position in the SELECT statement.

Executing the SQL SELECT statement generates a commit operation to the database by default. To avoid this, use the BeginTrans method on the session object before using the CreateDynaset method.

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.

Updatability Conditions

For the dynaset to be updatable, three conditions must be met:

- 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.
- 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 `Updatable` property of the dynaset 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.

Examples

This example demonstrates the `CreateCustomDynaset` method. Copy and paste this code into the definition section of a form, then press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

 'Create the OraDynaset Object using sliceSize as 256,perblock size as 16, no. of
 'blocks as 20, fetchLimit as 20,FetchSize as 4096

 Set OraDynaset = OraDatabase.CreateCustomDynaset("select empno, " & _
                  "ename from emp", 0&,256,16,20,20,4096)

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

End Sub
```

See Also: SnapShot Property on page 11-146

CreateDatabasePool Method

Applies To

OraSession Object on page 9-58

Description

Creates a pool of `OraDatabase` objects. Only one pool can be created for each `OraSession` object.

Usage

```
CreateDatabasePool (long initialSize, long maxSize, long timeoutValue, BSTR
database_name, BSTR connect_string, long options)
```

Arguments

The arguments for the method are:

Arguments	Description
initialSize	The initial size of the pool.
maxSize	The maximum size to which the pool can grow.
timeoutValue	If an `OraDatabase` object in the pool is idle for the *timeoutValue* value specified, the database connection that it contains is disconnected. The connection is reopened if the pool item is used again. This value is in seconds.
database_name	The Oracle network specifier used when connecting the data control to a database.
connectString	The user name 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 apply. "Constants" on page 10-212 shows the available modes.

Remarks

The `OpenDatabase` method of the `OraSession` object is used to establish a connection to an Oracle database. This method returns a reference to the `OraDatabase` object which is then used for executing SQL statements and PL/SQL blocks. The connection pool in OO4O is a pool of `OraDatabase` objects. The pool is created by invoking the `CreateDatabasePool` method of the `OraSession` interface.

Exceptions are raised by this call if:

- A pool already exists.
- An error occurs in creating a connection to Oracle Database.
- Invalid values for arguments are passed (that is, *initialSize* > *maxSize*).

The `LastServerErr` property of the `OraSession` object contains the code for the specific cause of the exception resulting from an Oracle Database error.

One possible connection error that could be returned is:

```
ORA-28001 "the password has expired"
```

The user can change the password using the `ChangePassword` method.

See Also:

- DestroyDatabasePool Method on page 10-128
- GetDatabaseFromPool Method on page 10-155
- RemoveFromPool Method on page 10-232
- ChangePassword (OraSession) Method on page 10-50
- LastServerErr Property on page 11-87

CreateDynaset Method

Applies To

OraDatabase Object on page 9-28

Description

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

Usage

```
Set oradynaset = oradatabase.CreateDynaset(sql_statement, options, SnapShotID)
Set oradynaset = oradatabase.DbCreateDynaset(sql_statement, options, SnapShotID)
```

Arguments

The arguments for the method are:

Arguments	Description
sql_statement	A `String` containing 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. Specifying the constant `ORADYN_DEFAULT` or the value `&H0&` gives the following defaults for the dynaset: ■ Behave like Visual Basic Mode for a database: Field values not explicitly set are set to `Null`, overriding database column defaults. ■ Perform automatic binding of database parameters. ■ Remove trailing blanks from character string data retrieved from the database. ■ Create an updatable dynaset. ■ Cache data on client. ■ Force a `MoveFirst` when the dynaset is created. ■ Maintain read-consistency.
SnapShotID [optional]	A `ID` of the snapshot obtained from the `SnapShot` property of an `OraDynaset` object.

Constants

The following table lists constants and values for the options flag.

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 remove trailing blanks from character string data retrieved from the database.
`ORADYN_READONLY`	`&H4&`	Force dynaset to be read-only.

Constant	Value	Description
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.
ORADYN_ORAMODE	&H10&	Behave the same as Oracle Mode for a database except affect only the dynaset being created. If database was created in Oracle Mode, the dynaset inherits the property from it (for compatibility).
ORADYN_NO_REFETCH	&H20&	Behave the same as ORADB_NO_REFETCH mode for a database except affect only the dynaset being created. If the database was created in ORADB_NO_REFETCH mode, the dynaset inherits the property for compatibility.
ORADYN_NO_MOVEFIRST	&H40&	Does not force a MoveFirst when the dynaset is created. BOF and EOF are both true.
ORADYN_DIRTY_WRITE	&H80&	Update and Delete methods do not check for read consistency.

These values can be found in the oraconst.txt file.

Remarks

Features such as simple views and synonyms can be used freely. You can also use schema references, column aliases, table joins, nested select statements and remote database references, but in each case, the dynaset is read-only.

If you use a complex expression or SQL function on a column, such as "sal + 100" or "abs(sal)", you get an updatable dynaset, but the column associated with the complex expression is not updatable.

Object names generally are not modifed, but in certain cases they can be changed. For example, if you use a column alias, you must use the alias to refer to the field by name. Also, if you use spaces in a complex expression, you must refer to the column without the spaces, since the database strips spaces. Note that you can always refer to a field by number, that is, by its ordinal position in the SELECT statement.

Executing the Update method generates a commit operation to the database by default. To avoid this, use the BeginTrans method on the session object before using the CreateDynaset method.

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, these conditions must be met:

- 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.
- Oracle Database 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 Updatable property of the dynaset 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.

The SnapShotID option causes a snapshot descriptor to be created for the SQLStmt object created. This property can later be obtained and used in creation of other SQLStmt or OraDynaset objects. Execution snapshots provide the ability to ensure that multiple commands executed in the context of multiple OraDatabase objects operate on the same consistent snapshot of the committed data in the database.

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

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

End Sub
```

See Also:

- Clone Method on page 10-52
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- MovePreviousn, MoveNextn, MoveRel, and MoveTo Methods on page 10-202
- OpenDatabase Method on page 10-212
- Updatable Property on page 11-171
- OraDynaset Object on page 9-30
- OraParameter Object on page 9-50
- OraParameters Collection on page 9-68
- Update Method on page 10-257
- BeginTrans Method on page 10-43
- SnapShot Property on page 11-146

CreateIterator Method

Applies To

OraCollection Object on page 9-19

Description

Creates an iterator to scan the elements of a collection.

Usage

```
OraCollection.CreateIterator
```

Remarks

This method creates an iterator for scanning the elements of an Oracle collection. Accessing collection elements using the iterator is faster than using an index on the instance of a collection.

Examples

Example: OraCollection Iterator

The following example illustrates the use of an Oracle collection iterator.

Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim CourseList As OraCollection
Dim Course As OraObject

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

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

'Create a dynaset object from division
Set OraDynaset = OraDatabase.CreateDynaset("select courses from" & _
                 "division where name='History'", 0&)

'Retrieve a Courses column from Division.
Set CourseList = OraDynaset.Fields("Courses").Value

'Create the iterator
CourseList.CreateIterator

'Initialize the iterator to point to the beginning of a collection
CourseList.InitIterator

'Call IterNext to read CourseList until the end
While CourseList.EOC = False
    Set Course = CourseList.ElementValue
```

```
    course_no = Course.course_no
    Title = Course.Title
    Credits = Course.Credits
    CourseList.IterNext
Wend

'Call IterPrev to read CourseList until the beginning
CourseList.IterPrev

While CourseList.BOC = False
    Set Course = CourseList.ElementValue
    course_no = Course.course_no
    Title = Course.Title
    Credits = Course.Credits
    CourseList.IterPrev
Wend
```

See Also:

- DeleteIterator Method on page 10-121
- InitIterator Method on page 10-171

CreateNamedSession Method

Applies To

OraSession Object on page 9-58

Description

Creates and returns a new named `OraSession` object.

Usage

```
orasession = orasession.CreateNamedSession(session_name)
```

Arguments

The arguments for the method are:

Arguments	Description
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 as long as the session object referred to is in scope. 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. Therefore, 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.

Examples

This example demonstrates the use of `ConnectSession` and `CreateNamedSession` methods 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
 Dim dfltsess As OraSession
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.CreateDynaset("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
```

See Also:

- CreateSession Method on page 10-109
- ConnectSession Method on page 10-69
- OraClient Object on page 9-18
- OraSessions Collection on page 9-69

CreateOraIntervalDS Method

Applies To

OraSession Object on page 9-58

Description

Creates the `OraIntervalDS` object. This `OraIntervalDS` represents an Oracle `INTERVAL DAY TO SECOND` data type.

Usage

```
Set OraIntervalDSObj = OraSession.CreateOraIntervalDS value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, an `OraIntervalDS`, or an `OraNumber` object.

Return Values

OraIntervalDS Object

Remarks

An `OraSession` object must be created before an `OraIntervalDS` object can be created.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If *value* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` represents.

A `Variant` of type `OraIntervalDS` can also be passed. A cloned `OraIntervalDS` is returned.

Examples

```
Dim oraIDS as OraIntervalDS
Dim oraIDS2 as OraIntervalDS
Dim oraNum  as OraNumber

'Create an OraIntervalDS using a string which represents 1 days, 2 hours,
'3 minutes, 4 seconds and 500000 nanoseconds
Set oraIDS = oo4oSession.CreateOraIntervalDS("1 2:3:4.005")

'Create an OraIntervalDS using a numeric value which represents
'1 days and 12 hours
Set oraIDS = oo4oSession.CreateOraIntervalDS(1.5)

'Create an OraIntervalDS using an OraIntervalDS
Set oraIDS2 = oo4oSession.CreateOraIntervalDS(oraIDS)
```

See Also:

- OraNumber Object on page 9-41
- OraIntervalDS Object on page 9-35

CreateOraIntervalYM Method

Applies To

OraSession Object on page 9-58

Description

Creates the `OraIntervalYM` object. This `OraIntervalYM` represents an Oracle `INTERVAL YEAR TO MONTH` data type.

Usage

```
Set OraIntervalYMObj = OraSession.CreateOraIntervalYM value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object.

Return Values

OraIntervalYM Object

Remarks

An `OraSession` object must be created before an `OraIntervalYM` object can be created.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] YEARS-MONTHS.

If *value* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

A `Variant` of type `OraIntervalYM` can also be passed. A cloned `OraIntervalYM` object is returned.

Examples

```
Dim oraIYM as OraIntervalYM
Dim oraIYM2 as OraIntervalYM

'Create an OraIntervalYM using a string which represents 1 year and 2 months
Set oraIYM = oo4oSession.CreateOraIntervalYM("1- 2")

'Create an OraIntervalYM using a numeric value which represents
'1 year and 6 months
Set oraIYM = oo4oSession.CreateOraIntervalYM(1.5)

'Create an OraIntervalYM using an OraIntervalYM
Set oraIYM2 = oo4oSession.CreateOraIntervalYM(oraIYM)
```

See Also:

- OraIntervalYM Object on page 9-37
- OraNumber Object on page 9-41

CreateOraNumber Method

Applies To

OraSession Object on page 9-58

Description

Creates an `OraNumber` object. This `OraNumber` represents an Oracle `NUMBER` data type.

Usage

```
OraNumber = OraSession.CreateOraNumber(inital_value, format)
```

Arguments

The arguments for the method are:

Arguments	Description
initial_value	Initial value of `OraNumber`. A `Variant` of type `OraNumber`, string or a numeric value.
format [optional]	Format string to be used when displaying `OraNumber` value.

Return Value

OraNumber Object

Remarks

For more information about format strings, see the format property on the `OraNumber` object.

See Also:

- ConnectSession Method on page 10-69
- CreateSession Method on page 10-109
- OraClient Object on page 9-18
- OraNumber Object on page 9-41
- OraSessions Collection on page 9-69

CreateOraObject (OraDatabase) Method

Applies To

OraDatabase Object on page 9-28

Description

Creates a value instance or referenceable object in the cache and returns the associated OO4O object.

Usage

```
OraObject1 = OraDatabase.CreateOraObject(schema_name)
OraRef1 = OraDatabase.CreateOraObject(schema_name,table_name)
OraCollection1 = OraDatabase.CreateOraObject(schema_name)
```

Arguments

The arguments for the method are:

Arguments	Description
OraObject1	A valid `OraObject` object representing a newly created value instance.
OraRef1	A valid `OraRef` object representing a newly created referenceable object.
OraCollection	A valid `OraCollection` object representing a newly created collection instance.
schema_name	A `String` specifying the schema name of the value instance to be created.
table_name	A `String` specifying the table name of the referenceable object to be created.

Remarks

If the *table_name* argument is not specified, it creates a value instance in the client and returns an `OraObject` or `OraCollection` object. If the *table_name* argument is specified, it creates a referenceable object in the database and returns an associated `OraRef` object.

Examples

`OraObject` and `OraRef` object examples are provided. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

Example: Creating an OraObject Object

The following example illustrates the use of the `CreateOraObject` method to insert a value instance. The row containing `ADDRESS` is inserted as a value instance in the database.

Dynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
```

```
Dim OraDynaset as OraDynaset
Dim AddressNew as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab", 0&)

' create a new Address object in OO4O
set AddressNew = OraDatabase.CreateOraObject("ADDRESS")

'initialize the Address object attribute to new value
AddressNew.Street = "Oracle Parkway"
AddressNew.State = "CA"

'start the dynaset AddNew operation and
'set the Address field to new address value
OraDynaset.Addnew
OraDynaset.Fields("ADDR").Value = AddressNew
OraDynaset.Update
```

OraParameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim AddressNew as OraObject

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

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

'create an  OraParameter object represent Address object bind Variable
OraDatabase.Parameters.Add "ADDRESS", Null, ORAPARM_INPUT, _
             ORATYPE_OBJECT, "ADDRESS"

' create a new Address object in OO4O
set AddressNew = OraDatabase.CreateOraObject("ADDRESS")

'initialize the Address object attribute to new value
AddressNew.Street = "Oracle Parkway"
AddressNew.State = "CA"

'set the Address to ADDRESS parameter
Oradatabase.Parameters("ADDRESS").Value = AddressNew

'execute the sql statement which updates Address in the person_tab
OraDatabase.ExecuteSQL ("insert into person_tab values ('Eric',30,:ADDRESS)")
```

Example: Creating an OraRef Object

The following example illustrates the use of the `CreateOraObject` method to insert referenceable objects.

In this example, a new PERSON is inserted as a referenceable object in the database.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim Person  as OraRef

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

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

'CreteOraObject   creates a new referenceable
'object in the PERSON_TAB object table and returns associated OraRef
set Person = OraDatabase.CreateOraObject("PERSON","PERSON_TAB")

'modify the attributes of Person
Person.Name = "Eric"

Person.Age = 35
'Update method inserts modified referenceable object in the PERSON_TAB.
Person.Update
```

CreateOraTimeStamp Method

Applies To

OraSession Object on page 9-58

Description

Creates a new OraTimeStamp object. This OraTimeStamp method represents an Oracle TIMESTAMP or an Oracle TIMESTAMP WITH LOCAL TIME ZONE data type.

Usage

```
Set OraTimeStampObj = OraSession.CreateOraTimeStamp value format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A Variant of type String, Date, or OraTimeStamp.
[in] [optional] *format*	TimeStamp format string to be used when displaying or interpreting an OraTimeStamp object as a string. If format is not specified, the TimeStamp string is interpreted using the session TIMESTAMP format (NLS_TIMESTAMP_FORMAT format).

Return Values

OraTimeStamp Object

Remarks

An OraSession object must created before an OraTimeStamp object can be created.

If *value* is a Variant of type String, the string format must match the datetime format specified in the *format* argument. If *format* is not specified, the string format must match the session TIMESTAMP format (NLS_TIMESTAMP_FORMAT).

If *format* is specified, it is stored in the Format property of the OraTimeStamp ; otherwise, the session TIMESTAMP format is stored in the OraTimeStamp Format property.

Examples

```
Dim oraTS as OraTimeStamp
Dim oraTS1 as OraTimeStamp
Dim date as Date

'Create an OraTimeStamp using a string assuming the session
'TIMESTAMP format is "DD-MON-RR HH.MI.SSXFF AM"
Set oraTS = oo4oSession.CreateOraTimeStamp("12-JAN-2003 12.0.0.0 PM")

'Create an OraTimeStamp using a string and a format
Set oraTS = oo4oSession.CreateOraTimeStamp("2003-01-12 12:00:00 PM", _
         "YYYY-MM-DD HH:MI:SS AM")

'Create an OraTimeStamp using a Date
date = #1/12/2003#
```

```
Set oraTS = oo4oSession.CreateOraTimeStamp(date)

'Create an OraTimeStamp  using an OraTimeStamp
Set oraTS1 = oo4oSession.CreateOraTimeStamp(oraTS)
```

See Also:

- OraTimeStamp Object on page 9-62
- OraNumber Object on page 9-41

CreateOraTimeStampTZ Method

Applies To

OraSession Object on page 9-58

Description

Creates a new OraTimeStampTZ object. This OraTimeStampTZ object represents an Oracle TIMESTAMP WITH TIME ZONE data type.

Usage

```
Set OraTimeStampTZObj = OraSession.CreateOraTimeStampTZ value format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A Variant of type String, Date, or OraTimeStampTZ.
[[in] [optional] *format*	TIMESTAMP WITH TIME ZONE format string to be used when displaying or interpreting an OraTimeStampTZ object as a string. If format is not specified, the TIMESTAMP WITH TIME ZONE string is interpreted using the session TIMESTAMP WITH TIME ZONE format (NLS_TIMESTAMP_TZ_FORMAT format).

Return Values

OraTimeStampTZ Object

Remarks

An OraSession object must be created before an OraTimeStampTZ object can be created.

If *value* is a Variant of type String, the string format must match the datetime format specified in the format argument if format is specified; otherwise, the string format must match the session TIMESTAMP WITH TIME ZONE format (NLS_TIMESTAMP_TZ_FORMAT).

If *value* is a Variant of type Date, the date-time value in the Date is interpreted as the date-time value in the time zone of the session. The TimeZone property in the OraTimeStampTZ object contains the time zone of the session.

If *format* is specified, it is stored in the Format property of the OraTimeStampTZ object, otherwise the session TIMESTAMP WITH TIME ZONE format is stored in the Format property of OraTimeStampTZ object.

Examples

```
Dim oraTSZ as OraTimeStampTZ
Dim oraTSZ1 as OraTimeStampTZ
Dim date as Date

'Create an OraTimeStampTZ using a string assuming the session
'TIMESTAMP WITH TIME ZONE format is "DD-MON-RR HH.MI.SSXFF AM TZH:TZM"
```

```
Set oraTSZ = oo4oSession.CreateOraTimeStampTZ( "12-JAN-2003" & _
             "12.0.0.0 PM -03:00")

'Create an OraTimeStampTZ using a string and a format
Set oraTSZ = oo4oSession.CreateOraTimeStampTZ( "2003-01-12" & _
             "12:00:00 PM -03:00", "YYYY-MM-DD HH:MI:SS AM TZH:TZM")

'Create an OraTimeStampTZ using a Date
date = #1/12/2003#
Set oraTSZ = oo4oSession.CreateOraTimeStampTZ(date)

'Create an OraTimeStampTZ  using an OraTimeStampTZ
Set oraTSZ1 = oo4oSession.CreateOraTimeStampTZ(oraTSZ)
```

See Also:

- OraTimeStampTZ Object on page 9-64
- OraNumber Object on page 9-41

CreatePLSQLCustomDynaset Method

Applies To

OraDatabase Object on page 9-28

Deprecated.

For information on how to perform these tasks, see "Returning PL/SQL Cursor Variables" on page 3-11.

Description

Creates a dynaset from a PL/SQL cursor using custom cache and fetch parameters. The SQL statement should be a stored procedure or anonymous block. The resulting dynaset is read-only. Attempting to set the SQL property results in an error. The dynaset can be refreshed with new parameters.

Usage

```
set OraDynaset = CreatePlsqlCustomDynaset(SQLStatement, CursorName, options,
slicesize, perblock, blocks, FetchLimit, FetchSize)
```

Arguments

The arguments for the method are:

Arguments	Description
SQLStatement	Any valid Oracle PL/SQL stored procedure or anonymous block.
CursorName	Name of the cursor created in the PL/SQL stored procedure.
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.
slicesize	Cache slice size.
perblock	Cache slices for each block.
blocks	Cache maximum number of blocks.
FetchLimit	Fetch array size.
FetchSize	Fetch array buffer size.

Constants

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 remove trailing blanks from character string data retrieved from the database.

Constant	Value	Description
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 use.
ORADYN_NO_MOVEFIRST	&H40&	Do not force a MoveFirst when the dynaset is created. BOF and EOF are both true.

These values can be found in the oraconst.txt file.

Remarks

The SQL statement must be a PL/SQL stored procedure with BEGIN and END around the call, as if it were executed as an anonymous PL/SQL block; otherwise, an error is returned. The *CursorName* argument should exactly match the cursor created inside the stored procedure or anonymous PL/SQL block; otherwise an error is returned. The cursor created inside the stored procedure should represent a valid SQL SELECT statement.

You do not need to bind the PL/SQL cursor variable using the OraParameters Add method if the stored procedure returns a cursor as an output parameter. You can still use PL/SQL bind variables in conjunction with the OraParameters collection.

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

Specifying ORADYN_READONLY, ORADYN_ORAMODE, ORADYN_NO_REFETCH, ORADYN_DIRTY_WRITE options have no effect on the dynaset creation.

See Also:

- OraParameters Collection on page 9-68
- Add Method on page 10-8

CreatePLSQLDynaset Method

Applies To

OraDatabase Object on page 9-28

Deprecated.

For information on how to perform these tasks, see "Returning PL/SQL Cursor Variables" on page 3-11.

Description

Creates a dynaset from a PL/SQL cursor. The SQL statement should be a stored procedure or an anonymous block. The resulting dynaset is read-only and attempting to set SQL property results in an error. Dynasets can be refreshed with new parameters similar to dynasets without cursors.

Usage

```
set OraDynaset = CreatePLSQLDynaset(SQLStatement, CursorName, options)
```

Arguments

Arguments	Description
SQLStatement	Any valid Oracle PL/SQL stored procedure or anonymous block.
CursorName	Name of the cursor created in the PL/SQL stored procedure.
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.

Constants

The options flag values are:

Constant	Value	Description
ORADYN_DEFAULT	&H0&	Accept the default behavior.
ORADYN_NO_BLANKSTRIP	&H2&	Do not remove trailing blanks from character string data retrieved from the database.
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.
ORADYN_NO_MOVEFIRST	&H40&	Do not force a MoveFirst operation when the dynaset is created. BOF and EOF are both true.

These values can be found in the oraconst.txt file.

Remarks

The SQL statement must be a PL/SQL stored procedure with BEGIN and END statements around the call, as if it were executed as an anonymous PL/SQL block; otherwise an error is returned. The *CursorName* argument should exactly match the cursor created inside the stored procedure or anonymous PL/SQL block; otherwise, an error is returned. Cursors created inside the stored procedure should represent a valid SQL SELECT statement.

You do not need to bind the PL/SQL cursor variable using the OraParameters.Add method if the stored procedure returns a cursor as a output parameter. You can still use PL/SQL bind variables in conjunction with the OraParameters collection.

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

Specifying the ORADYN_READONLY, ORADYN_ORAMODE, ORADYN_NO_REFETCH, or ORADYN_DIRTY_WRITE options have no effect on the dynaset creation.

Examples

This example demonstrates the use of PL/SQL cursor in the CreatePlsqlDynaset method and Refresh method. This example returns a PL/SQL cursor as a dynaset for the different values of the DEPTNO parameter. Make sure that corresponding stored procedure (found in EMPCUR.SQL) is available in the Oracle database. and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase

 Dim OraDynaset As OraDynaset

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

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

' Create the Deptno parameter
 OraDatabase.Parameters.Add "DEPTNO", 10, ORAPARM_INPUT
 OraDatabase.Parameters("DEPTNO").ServerType = ORATYPE_NUMBER

' Create OraDynaset based on "EmpCursor" created in stored procedure.
 Set OraDynaset = OraDatabase.CreatePLSQLDynaset("Begin Employee.GetEmpData" & _
               "(:DEPTNO,:EmpCursor); end;", "EmpCursor", 0&)

 'Should display KING
 MsgBox OraDynaset.Fields("ENAME").Value

 'Should display 7839
 MsgBox OraDynaset.Fields("EMPNO").Value

 ' Now set the deptno value to 20
 OraDatabase.Parameters("DEPTNO").Value = 20

 'Refresh the dynaset
 OraDynaset.Refresh

 'Should display JONES
```

```
MsgBox OraDynaset.Fields("ENAME").Value

'Should display 7566
MsgBox OraDynaset.Fields("EMPNO").Value

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

End Sub
```

CreateSession Method

Applies To

OraClient Object on page 9-18

Description

Creates a new named `OraSession` object.

Usage

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

Arguments

The arguments for the method are:

Arguments	Description
session_name	A `String` specifying the name of the session.

Remarks

Use this method to 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.

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraClient As OraClient
 Dim OraSession As OraSession
 Dim NamedOraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.CreateDynaset("select * from emp", 0&)

End Sub
```

See Also: OraSession Object on page 9-58

CreateSQL Method

Applies To

OraDatabase Object on page 9-28

Description

Executes the SQL statement and creates an `OraSQLStmt` object from the specified SQL statement and options.

Usage

```
Set orasqlstmt = oradatabase.CreateSQL(sql_statement, options)
```

Arguments

The arguments for the method are:

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

Constants

The options flag values are:

Constant	Value	Description
ORASQL_NO_AUTOBIND	&H1&	Do not perform automatic binding of database parameters.
ORASQL_FAILEXEC	&H2&	Raise error and do not create SQL statement object.
ORASQL_NONBLK	&H4&	Execute SQL in a nonblocking state.

These values can be found in the `oraconst.txt` file.

Remarks

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.

Executing the SQL statement generates a commit to the database by default. To avoid this, use the `BeginTrans` method on the session object before using the `CreateSQL` method.

When executing PL/SQL blocks or calling stored procedures, you must include a `BEGIN` and `END` statement 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.

If the ORASQL_FAILEXEC option is used, an error is raised during SQLstmt object creation failure (on SQLstmt object refresh). The SQLstmt object is not created and cannot be refreshed.

> **Note:** Use the CreateSQL method with care, because any SQL statement or PL/SQL block that is executed might cause errors afterward when you use the Edit method on open dynasets.

Data Type

String

Examples

This example demonstrates the use of parameters, the CreateSQL method, the Refresh method, and the SQL property for OraSQLStmt object. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraSqlStmt As OraSQLStmt

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

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

 OraDatabase.Parameters.Add "EMPNO", 7369, 1
 OraDatabase.Parameters("EMPNO").ServerType = 2  'ORATYPE_NUMBER

 OraDatabase.Parameters.Add "ENAME", 0, 2
 OraDatabase.Parameters("ENAME").ServerType = 1  'ORATYPE_VARCHAR2

 Set OraSqlStmt = OraDatabase.CreateSQL("Begin Employee.GetEmpName" & _
          "(:EMPNO, :ENAME); end;", 0&)

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

 'Should display SMITH
 MsgBox OraDatabase.Parameters("ENAME").Value

 'Change the value of the empno parameter.
 OraDatabase.Parameters("EMPNO").Value = 7499

 'Refresh the sqlstmt
 OraSqlStmt.Refresh

 'Should display ALLEN
 MsgBox OraDatabase.Parameters("ENAME").Value

 'Notice that the SQL statement is NOT modified.
```

```
MsgBox OraSqlStmt.SQL

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

End Sub
```

See Also:

- "Asynchronous Processing" on page 3-16 for more information about the ORASQL_NONBLK option
- BeginTrans Method on page 10-43
- OraSQLStmt Object on page 9-60
- CreateSQL Method on page 10-111
- ExecuteSQL Method on page 10-144
- Refresh Method on page 10-225

CreateTempBLOB/CLOB Method

Applies To

OraDatabase Object on page 9-28

Description

Creates a temporary LOB in the database.

Usage

```
Set OraBLOB = OraDatabase.CreateTempBLOB(use_caching)
Set OraCLOB = OraDatabase.CreateTempCLOB(use_caching)
```

Arguments

The arguments for the method are:

Arguments	Description
use_caching	A boolean value that specifies whether Oracle Database uses caching when accessing this LOB. The default value is `False`.

Remarks

Temporary LOBs are LOBs that do not exist permanently in the database. OO4O programmers commonly use temporary LOBs to pass into stored procedures and functions that have LOB arguments.

Temporary LOBs do not require or take part in transactions. (It is not necessary to acquire a lock before write operations, and rollbacks have no effect on temporary LOBs.)

The *use_caching* argument directs Oracle to use caching when accessing the temporary LOB. This is suggested when multiple accesses are expected on a single LOB. Caching is not required for the typical case, where a LOB is created, filled with data, passed to a stored procedure, and then discarded.

Temporary LOBs exist on the database until no more references to the corresponding `OraBLOB` or `OraCLOB` exist on the client. Note that these references include any `OraParameter` or `OraParamArray` that contain a temporary `OraBLOB` or `OraCLOB` object.

Examples

Example: Passing a Temporary CLOB to a Stored Procedure

The following example illustrates the use of the `CreateTempClob` method to create a `OraCLOB`. The `OraCLOB` is then populated with data and passed to a stored procedure which has an argument of type `CLOB`.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraClob as OraClob

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

```
'Create the OraDatabase Object by opening a connection to Oracle.

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

'Create the stored procedure used in this example
OraDatabase.ExecuteSQL ("create or replace procedure GetClobSize" & _
           "(in_clob IN CLOB, clobsize OUT NUMBER) as Begin clobsize" & _
           " := DBMS_LOB.GETLENGTH(in_clob); End;")

'create an OraParameter object to represent Clob bind Variable
OraDatabase.Parameters.Add "CLOB", Null, ORAPARM_INPUT,  ORATYPE_CLOB

'the size will go into this bind variable
OraDatabase.Parameters.Add "CLOBSIZE", Null, ORAPARM_OUTPUT,  ORATYPE_NUMBER

' create a temporary CLOB
set OraClob = OraDatabase.CreateTempClob

'Populate the OraClob with some data. Note that no row locks are needed.
OraClob.Write "This is some test data"

'set the Parameter Value to the temporary Lob
OraDatabase.Parameters("CLOB").Value = OraClob

'execute the sql statement which updates Address in the person_tab
OraDatabase.ExecuteSQL ("Begin GetClobSize(:CLOB, :CLOBSIZE); end;")

'Display the size
MsgBox OraDatabase.Parameters("CLOBSize").Value

'these two lines force the temporary clob to be freed immediately
OraDatabase.Parameters.Remove "CLOB"
Set OraClob = nothing
```

Delete Method

Applies To

OraDynaset Object on page 9-30

Description

Deletes the current row of the specified dynaset.

Usage

```
oradynaset.Delete
oradynaset.DbDelete
```

Remarks

A row must be current before you can use the `Delete` method; otherwise, an error occurs.

Note that after you call the `Delete` method on a given row in a dynaset in a global transaction (that is, once you issue a `BeginTrans` method), locks remain on the selected rows until you call a `CommitTrans` or `Rollback` method.

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 movement occurs, you cannot make the deleted row current again.

You cannot restore deleted records except by using transactions.

> **Note:** A call to an `Edit`, `AddNew`, or `Delete` method, cancels any outstanding `Edit` or `AddNew` calls before proceeding. Any outstanding changes not saved using an `Update` method are lost during the cancellation.

Examples

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

```
Sub Form_Load ()

'Declare variables
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset

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

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

'Create the OraDynaset Object. Only select the employees in Department 10.
Set OraDynaset = OraDatabase.CreateDynaset("select * from emp where" & _
```

```
          "deptno=10", 0&)

 Do Until OraDynaset.EOF
   OraDynaset.Delete
   OraDynaset.MoveNext
 Loop
 MsgBox "All employees from department 10 removed."

End Sub
```

See Also:

- AddNew Method on page 10-21
- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- Edit Method on page 10-134
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235

Delete (OraCollection) Method

Applies To

OraCollection Object on page 9-19

Description

Deletes an element at given index. This method is available only in an `OraCollection` of type `ORATYPE_TABLE` (nested table).

Usage

```
OraCollection.Delete index
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *index*	An `Integer` specifying the index of the element to be deleted.

Remarks

The `Delete` method creates *holes* in the client-side nested table. This method returns an error if the element at the given index has already been deleted or if the given index is not valid for the given table.

Examples

The following example illustrates the `Delete` method. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim CourseList as OraCollection

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

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

'create a dynaset object from division
set OraDynaset = OraDatabase.CreateDynaset("select * from division", 0&)

'retrieve a Courses column from Division.
'Here Value property of OraField object returns CourseList OraCollection
set CourseList = OraDynaset.Fields("Courses").Value

'Delete the CourseList  NestedTable at index 2.
'Before that lock should be obtained
OraDynaset.Edit
CourseList.Delete 2
```

```
OraDynaset.Update
```

See Also: Type (OraCollection) Property on page 11-167

Delete (OraRef) Method

Applies To

OraRef Object on page 9-52

Description

Deletes a referenceable object in the database.

Usage

```
OraRef.Delete
```

Remarks

Accessing attributes on the deleted instance results in an error.

Examples

The following example illustrates the `Delete` method. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim Person  as OraRef

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

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

'create an  OraParameter object represent Person object bind Variable
OraDatabase.Parameters.Add "PERSON", Null, ORAPARM_OUTPUT, ORATYPE_REF,"PERSON"

'execute the sql statement which selects person
'from the customers table for account = 10
OraDatabase.ExecuteSQL ("BEGIN select aperson into :PERSON from customers" & _
                   "where account = 10;  END;")

'get the Person object from OraParameter
set Person = OraDatabase.Parameters("PERSON").Value

'delete the Person object in the server for modifying its attributes
Person.Delete
```

DeleteIterator Method

Applies To

OraCollection Object on page 9-19

Description

Deletes a collection iterator.

Usage

```
OraCollection.DeleteIterator
```

Remarks

None.

Examples

See "Example: OraCollection Iterator" on page 10-88

See Also: "CreateIterator Method" on page 10-88

Dequeue (OraAQ) Method

Applies To

OraAQ Object on page 9-3

Description

Dequeues a message.

Usage

```
Q.Dequeue()
```

Remarks

The message attributes can be accessed with the `OraAQMsg` interface contained in this object. On success, this method returns the message identifier as an array of bytes. Otherwise, it returns an empty array (null).

Examples

Note: The following code sample are models for dequeuing messages.

A complete AQ sample can be found in the `\OO4O\VB\SAMPLES\AQ` directory.

Example: Dequeuing Messages of RAW Type

```
 'Dequeue the first message available
Q.Dequeue
Set Msg = Q.QMsg

'Display the message content
MsgBox Msg.Value

'Dequeue the first message available without removing it
' from the queue
Q.DequeueMode = ORAAQ_DQ_BROWSE

'Dequeue the first message with the correlation identifier
' equal to "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_FIRST_MSG
Q.correlate = "RELATIVE_MESSAGE_ID"
Q.Dequeue

'Dequeue the next message with the correlation identifier

' of "RELATIVE_MSG_ID"
Q.Navigation = ORAAQ_DQ_NEXT_MSG
Q.Dequeue

'Dequeue the first high priority message
Msg.Priority = ORAQMSG_HIGH_PRIORITY
```

```
Q.Dequeue

'Dequeue the message enqueued with message id of Msgid_1
Q.DequeueMsgid = Msgid_1
Q.Dequeue

'Dequeue the message meant for the consumer "ANDY"
Q.consumer = "ANDY"
Q.Dequeue

'Return immediately if there is no message on the queue
Q.wait = ORAAQ_DQ_NOWAIT
Q.Dequeue
```

Example: Dequeuing Messages of Oracle Object Types

```
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")
Set QMsg = Q.AQMsg(23, "MESSAGE_TYPE","SCOTT")

'Dequeue the first message available without removing it
Q.Dequeue
OraObj = QMsg.Value

'Display the subject and data
MsgBox OraObj("subject").Value & OraObj("Data").Value
```

Describe Method

Applies To

OraDatabase Object on page 9-28

Description

Describes a schema object. This method returns an instance of the `OraMetaData` interface.

Usage

```
OraMetaDataObj = OraDatabase.Describe(SchemaObjectName)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *SchemaObjectName*	A `String` representing the name of the schema object to be described.

Remarks

The following schema object types can be described:

- `Tables`
- `Views`
- `Procedures`
- `Functions`
- `Packages`
- `Sequences`
- `Collections` (`VARRAY`s or nested tables)
- `Types`

Describing any other schema object (for example, a column) or an invalid schema object name raises an error. You should navigate to schema objects not listed here, rather than describing them directly.

This method takes the name of a schema object, such as `emp`, and returns a COM Automation object (`OraMetaData`). The `OraMetaData` object provides methods for dynamically navigating and accessing all the attributes (`OraMDAttribute` collection) of a schema object described.

Examples

Simple Describe Example

The following Visual Basic code illustrates a how to use the `Describe` method to retrieve and display several attributes of the `emp` table.

```
Set emp = OraDatabase.Describe("emp")
```

```
'Display the name of the Tablespace
MsgBox emp!tablespace
'Display name and data type of each column in the emp table.
Set empColumns = emp!ColumnList
Set ColumnList = empColumns.Value

for i = 0 to ColumnList.Count - 1
  Set Column = ColumnList(i).Value
  MsgBox "Column: " & Column!Name & " Data Type: " & Column!Data Type
Next i
```

Describing a Table Example

Before running the following example, make sure that you have the necessary datatypes and tables in the database. See "Schema Objects Used in OraMetaData Examples" on page A-3.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim OraMetaData As OraMetaData
Dim OraMDAttribute As OraMDAttribute
Dim ColumnList As OraMetaData
Dim Column As OraMetaData

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

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

'Use Describe to retrieve the metadata object
Set OraMetaData = OraDatabase.Describe("EMP")

'Display the type of the metadata
MsgBox TypeofMetaData & OraMetaData.Type

'Display the count of attributes belonging to the table
MsgBox NumberOfAttributes & OraMetaData.Count

'Attribute can be accessed using the explicit OraMetaData property: Attribute
'The index can be an integer or the attribute name
Set OraMDAttribute = OraMetaData.Attribute(0)
MsgBox "ObjectID: " & OraMDAttribute.Value

'Since Attribute is the default property of OraMetaData, an attribute can
' be accessed as follows. Here, we use attribute name as an index
Set OraMDAttribute = OraMetaData("ObjectID")
MsgBox "Name: " & OraMDAttribute.Name
MsgBox "Value: " & OraMDAttribute.Value

'Value is the default property of OraMDAttribute, the following shows
'the Value of property "IsClustered" for the table
MsgBox "Is Clustered: " & OraMetaData!IsClustered
MsgBox "Is Partitioned: " & OraMetaData!IsPartitioned

'Retrieve the Column List
Set OraMDAttribute = OraMetaData!ColumnList
```

```
' Use IsMDObject property to check whether an attribute's value is an OraMetaData
If (OraMDAttribute.IsMDObject()) Then
       Set ColumnList = OraMDAttribute.Value
      'Display the name and data type of each column
       For I = 0 To ColumnList.Count - 1
        Set Column = ColumnList(I).Value

' Each column is again an OraMetaData
    MsgBox "Column: " & Column!Name & " data type: " & Column!Data Type
  Next I
End If
```

Example: Describing a User-Defined Type

Before running the following example, make sure that you have the necessary datatypes and tables in the database. See "Schema Objects Used in OraMetaData Examples" on page A-3.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim OraMetaData As OraMetaData
Dim OraMDAttribute As OraMDAttribute
Dim attrList As OraMetaData
Dim attr As OraMetaData

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

'Create the OraDatabase Object by opening a connection to Oracle.
Set OraDatabase = OraSession.OpenDatabase("ExampleDB", "scott/tiger",0&)
Set OraMetaData = OraDatabase.Describe("ORAMD_ADDRESS")
NumAttributes = OraMetaData!NumAttributes
NumMethods = OraMetaData!NumMethods
MsgBox "The Address type has " & NumAttributes & " attributes"
MsgBox "Address Object has " & NumMethods & " methods"

'Retrieve the attribute list of this type object
Set attrList = OraMetaData!Attributes.Value

'Display the name and data type of each attribute
For I = 0 To attrList.Count - 1
  Set attr = attrList(I).Value
  ' each attr is actually an OraMetaData
  MsgBox "Attribute Name: " & attr!Name
  MsgBox "Attribute Type: " & attr!TypeName

Next I
```

Example: Describing Unknown Schema Objects

Before running the following example, make sure that you have the necessary datatypes and tables in the database. See "Schema Objects Used in OraMetaData Examples" on page A-3.

```
Sub RecursiveDescribe(name$, xMD As OraMetaData)

Dim xMDAttr As OraMDAttribute
For I = 0 To xMD.Count - 1
    Set xMDAttr = xMD.Attribute(I)

    ' If an attribute can be described further, describe it,
```

```
    ' otherwise display its attribute name & value
    If (xMDAttr.IsMDObject) Then
        RecursiveDescribe xMDAttr.name, xMDAttr.Value
    Else
        MsgBox name & "->" & xMDAttr.name & " = " & xMDAttr.Value
  End If
Next I

End Sub
Sub Main()

'This example displays all the attributes of any schema object given
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim xMD As OraMetaData
Dim x As String

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

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

' x is any database object, here the EMP table is used as an example
x = "EMP"
Set xMD = OraDatabase.Describe(x)
MsgBox x & " is of the type " & xMD.Type
RecursiveDescribe x, xMD

End Sub
```

See Also:

- OraMetaData Object on page 9-39
- OraMDAttribute Object on page 9-38

DestroyDatabasePool Method

Applies To

OraSession Object on page 9-58

Description

The pool is implicitly destroyed if its parent session object is destroyed. It can also be destroyed at any time by invoking the `DestroyDatabasePool` method.

Usage

```
DestroyDatabasePool()
```

Remarks

An exception is raised by this call if the pool does not exist.

See Also: CreateDatabasePool Method on page 10-83

DisableBuffering (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Disables buffering of LOB operations.

Usage

```
OraBlob.DisableBuffering
OraClob.DisableBuffering
```

Remarks

This method does not automatically flush the buffers. The `FlushBuffer` method should be used to flush any changes before buffering is disabled.

See Also:

- EnableBuffering (OraLOB) Method on page 10-139
- FlushBuffer (OraLOB) Method on page 10-154

Div (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Divides the `OraIntervalDS` object by a divisor.

Usage

```
OraIntervalDSObj.Div divisor
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *divisor*	A `Variant` for type numeric value or an `OraNumber` object to be used as the divisor.

Remarks

The result of the operation is stored in the `OraIntervalDS` object, overwriting any previous value. There is no return value.

Div (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Divides the `OraIntervalYM` object by a divisor.

Usage

```
OraIntervalYMObj.Div divisor
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *divisor*	A `Variant` for type numeric value or an `OraNumber` object to be used as the divisor.

Remarks

The result of the operation is stored in the `OraIntervalYM` object, overwriting any previous value. There is no return value.

Div (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Divides an `OraNumber` object by a numeric argument.

Usage

```
OraNumber.Div operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber` object, or a numeric value.

Remarks

The result of the operation is stored in an `OraNumber` object . There is no return value.

The *operand* must not be equal to zero, or a divide by zero error is raised.

DynasetCacheParams Method

Applies To

OraParameter Object on page 9-50

Description

Specifies the dynaset cache and fetch parameters for the dynaset created from the PL/SQL cursor.

Usage

```
oraparameter.DynasetCacheParams SliceSize,perblock, Blocks, FetchLimit,FetchSize
```

Arguments

The arguments for the method are:

Arguments	Description
SliceSize	Cache slice size.
perblock	Cache slices for each block.
Blocks	Cache maximum number of blocks.
FetchLimit	Fetch array size.
FetchSize	Fetch array buffer size.

Remarks

This method should be called before executing the PL/SQL procedure containing a cursor variable. By default, the dynaset is created with default cache and fetch parameters specified in the registry.

Edit Method

Applies To

OraDynaset Object on page 9-30

Description

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

Usage

```
oradynaset.Edit
oradynaset.DbEdit
```

Remarks

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

> **Note:** The cached data is not compared to the database with `BLOB` and `CLOB`, `Object`, `REF`, and collection types, and the data is updated regardless (dirty writes).

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 an `Edit`, `AddNew`, or `Delete` method cancels any outstanding `Edit` or `AddNew` calls before proceeding. Any outstanding changes not saved using an `Update` operation are lost during the cancellation.

Examples

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

```
Sub Form_Load ()

'Declare variables
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
```

```
Dim OraDynaset As OraDynaset

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

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

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

 'Traverse until EOF is reached, settingeach employee's salary to zero
 Do Until OraDynaset.EOF
   OraDynaset.Edit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.Update
   OraDynaset.MoveNext
 Loop
 MsgBox "All salaries set to ZERO."

End Sub
```

See Also:

- AddNew Method on page 10-21
- CreateDynaset Method on page 10-85
- Delete Method on page 10-116
- OpenDatabase Method on page 10-212
- Update Method on page 10-257

Edit (OraRef) Method

Applies To

OraRef Object on page 9-52

Description

Locks a referenceable object in the database.

Usage

```
OraRef.Edit
```

Remarks

Call this method before modifying any attributes of an underlying referenceable object of `OraRef` or an error is raised. This call makes a network round-trip to lock the object in the database. An error is raised if the object is changed by another user in the database. The object can also be locked during the pin operation using the `EditOption` property.

Examples

The following examples update the attributes of the `"PERSON"` referenceable object in the database.

Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

Dynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)

'retrieve a aperson column from customers.
'Here Value property of OraField object 'returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'locks the Person object in the server for modifying its attributes
Person.Edit
  Person.Name = "Eric"
  Person.Age = 35
'Update method flushes the modified referenceable object in the server
Person.Update
```

Parameter Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim Person as OraRef

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

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

'create an  OraParameter object represent Address object bind Variable
OraDatabase.Parameters.Add "PERSON", Null, ORAPARM_OUTPUT, _
                ORATYPE_REF,"PERSON"

'execute the sql statement which selects person from the customers table
OraDatabase.ExecuteSQL ("BEGIN select aperson into :PERSON" & _
                "from customers where account = 10;  END;")

'get the Person object from OraParameter
set Person = OraDatabase.Parameters("PERSON").Value

'locks the Person object in the server for modifying its attributes
Person.Edit
  Person.Name = "Eric"
  Person.Age = 35

'Update method flushes the modified referenceable object in the server
Person.Update
```

See Also: EditOption (OraRef) Property on page 11-52

ElementValue Method

Applies To

OraCollection Object on page 9-19

Description

Returns the current value of the collection element to which the iterator points.

Usage

```
elem_val = OraCollection.ElementValue
```

Arguments

The arguments for the method are:

Arguments	Description
elem_val	A `Variant` representing element value of the collection.

ElementType

For elements of type `Object` and `REF`, element values are returned as corresponding OO4O objects for that type. The following table shows the element type and return value of the elements:

ElementType	Element Value
`Object`	`OraObject`
`REF`	`OraRef`
`Date`	`String`
`Number`	`String`
`CHAR,VARCHAR2`	`String`
`Real`	`Real`
`Integer`	`Integer`

Remarks

Calling this method when the `EOC` or `BOC` property returns `True` raises an error. The `Variant` type of the element depends on the element type of the collection.

Examples

See "Example: OraCollection Iterator" on page 10-88

See Also:

- CreateIterator Method on page 10-88
- IterNext Method on page 10-187
- IterPrev Method on page 10-188

EnableBuffering (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Enables buffering of LOB operations.

Usage

```
OraBlob.EnableBuffering
OraClob.EnableBuffering
```

Remarks

When enabled, buffering uses the LOB Buffering subsystem to minimize network round-trips by buffering changes until the `FlushBuffer` method is called. This can be beneficial to applications that perform a series of repeated small reads and writes to specific areas of a LOB.

There are many caveats and restrictions for using LOB buffering. These are summarized here, but for complete information, see the *Oracle Database SecureFiles and Large Objects Developer's Guide*.

Restrictions

- The following LOB methods cannot be used while buffering is enabled:
  - `Append`
  - `Copy`
  - `Erase`
  - `Size`
  - `Trim`
  - `CopyFromBFILE`
  - `CopyFromFile`
  - `CopyToFile`
- There is currently a 512 KB limit to the amount of a single read/write operation.
- Error reporting for buffered operations is delayed until the next database access.
- Transactional support is not guaranteed. Users must roll back changes manually if an error occurs.
- Do not perform updates to a LOB column that bypasses the buffering system while in the same transaction as a buffer-enabled LOB. Performing an `INSERT` statement can cause this.
- Only one LOB object is allowed to perform buffered writes to a given LOB. Other LOB objects that point to the same LOB raise an error if they attempt a buffered write.
- A LOB object taken from an `OraParameter` object raises an error if it is buffer-enabled and bound to an `OUT` parameter.

- The `Clone` method can raise an error for buffer enabled LOBs.
- Appending directly to the end of the LOB is allowed, but any write operation whose offset extends beyond the end of the LOB and results in blank padding (for `CLOB`) or zero padding (for `BLOB`) raises an error.

See Also: *Oracle Database SecureFiles and Large Objects Developer's Guide*

Enqueue (OraAQ) Method

Applies To

OraAQ Object on page 9-3

Description

Enqueues the message (`OraAQMsg`) contained in this object.

Usage

```
Msgid = Q.Enqueue
```

Remarks

On success, this method returns the message identifier as an array of bytes. Otherwise, it returns an empty array (null).

Examples

Note: The following code samples are models for enqueuing messages, but cannot be run as is.

A complete AQ sample can be found in the `\OO4O\VB\SAMPLES\AQ` directory.

Enqueuing Messages of Type RAW

```
'Create an OraAQ object for the queue "DBQ"
Dim Q as OraAQ
Dim Msg as OraAQMsg
Dim OraSession as OraSession
Dim DB as OraDatabase

Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set DB = OraSession.OpenDatabase("mydb", "scott/tiger" 0&)
Set Q = DB.CreateAQ("DBQ")

'Get a reference to the AQMsg object
Set Msg = Q.AQMsg
Msg.Value = "Enqueue the first message to a RAW queue."

'Enqueue the message
Q.Enqueue

'Enqueue another message.
Msg.Value = "Another message"
Q.Enqueue

'Enqueue a message with non-default properties.
Msg.Priority = ORAQMSG_HIGH_PRIORITY
Msg.Delay = 5
Msg.Value = "Urgent message"
Q.Enqueue
Msg.Value = "The visibility option used in the enqueue call" & _
```

```
           "is ORAAQ_ENQ_IMMEDIATE"
Q.Visible = ORAAQ_ENQ_IMMEDIATE
Msgid = Q.Enqueue

'Enqueue Ahead of message Msgid_1
Msg.Value = "First Message to test Relative Message id"
Msg.Correlation = "RELATIVE_MESSAGE_ID"

Msg.delay = ORAAQ_MSG_NO_DELAY
Msgid_1 = Q.Enqueue
Msg.Value = "Second message to test RELATIVE_MESSAGE_ID is queued" & _
            " ahead of the First Message "
Q.RelMsgId = Msgid_1
Msgid = Q.Enqueue
```

Enqueuing Messages of Oracle Object Types

```
'Prepare the message. MESSAGE_TYPE is a user defined type in the "AQ" schema
Set OraMsg = Q.AQMsg(23, "MESSAGE_TYPE","SCOTT")
Set OraObj = DB.CreateOraObject("MESSAGE_TYPE")

OraObj("subject").Value = "Greetings from OO4O"
OraObj("text").Value = "Text of a message originated from OO4O"

Msgid = Q.Enqueue
```

Erase (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Erases the specified portion of the LOB value of this object starting at the specified offset.

Usage

```
OraBlob.Erase amount, offset
OraClob.Erase amount, offset
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *amount*	An Integer specifying the maximum number of characters or bytes to be erased.
[in] *offset* [optional]	An Integer specifying absolute offset of the LOB value from which to start erasing. Default value is 1.

Remarks

Obtain either a row-level lock or object-level lock before calling this method. The actual number of characters or bytes and the requested number differ if the end of the LOB value is reached before erasing the requested number of characters or bytes. For BLOB types, erasing means that zero-byte fillers overwrite the existing LOB value. For CLOB types, erasing means that spaces overwrite the existing LOB value.

ExecuteSQL Method

Applies To

OraDatabase Object on page 9-28

Description

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

Usage

```
rowcount = oradatabase.ExecuteSQL(sql_statement)
rowcount = oradatabase.DbExecuteSQL(sql_statement)
```

Arguments

The arguments for the method are:

Arguments	Description
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* argument 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.

Executing the SQL statement generates a commit to the database by default. To avoid this, use the BeginTrans method on the session object before using the ExecuteSQL method.

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 statement 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:** The ExecuteSQL method should be used with care because any SQL statement or PL/SQL block that is executed can adversely affect open dynasets. This is 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 a SQL commit statement, 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. See the *Oracle Database SQL Language Reference* for more details about DCL, DDL, and transactions.

Data Type

`Long Integer`

Examples

Example: ExecuteSQL

This example uses the `Add` and `Remove` parameter methods, the `ServerType` parameter property, and the `ExecuteSQL` database method to call the stored procedure `GetEmpName` and the stored function `GetSal`. Before running the example, run the `ORAEXAMP.SQL` file to create `GetEmpName` and `GetSal` as well as other necessary object types and LOBs in Oracle Database. Then, copy and paste this OO4O code example into the definition section of a form and run the program.

```
Sub Form_Load ()

'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDatabase

 '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.ExecuteSQL ("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.ExecuteSQL ("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
```

See Also:

- *Oracle Database SQL Language Reference*
- CreateDynaset Method on page 10-85
- OraParameters Collection on page 9-68
- Transactions Property on page 11-162

Exist (OraCollection) Method

Applies To

OraCollection Object on page 9-19

Description

Returns `True` if an element exists at a given index; otherwise, returns. Valid only for `OraCollection` of Type `ORATYPE_TABLE`.

Usage

```
exists = OraCollection.Exist index
```

Arguments

The arguments for the method are:

Arguments	Description
[out] *exists*	A Boolean value specifying the existence status of the element.
[in] *index*	An `Integer` specifying the index of the element.

Remarks

None.

Exp (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates *e* to the power of an `OraNumber` object.

Usage

```
OraNumber.Exp
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

FetchOraRef Method

Applies To

OraDatabase Object on page 9-28

Description

Fetches a referenceable object into the cache and returns the associated `OraRef` object.

Usage

```
Set OraRef = OraDatabase.FetchOraRef(hex_value)
```

Arguments

The arguments for the method are:

Arguments	Description
hex_value	A `String` containing the hexadecimal value of the `REF`.

Remarks

The *hex_value* argument can be obtained through the `OraRef.HexValue` property or from an XML document generated by the `OraDynaset.GetXML` method.

See Also:

- HexValue (OraRef) Property on page 11-73
- GetXML Method on page 10-163

FieldSize Method

Applies To

OraField Object on page 9-33

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( )
```

Remarks

Returns the number of bytes stored in a `LONG` or `LONG RAW` field, up to a value of around 64 KB. If the field contains more than 64 KB, then the `FieldSize` method returns `-1`.

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

Data Type

Long Integer

See Also:

- AppendChunk Method on page 10-28
- GetChunk Method on page 10-156
- OraField Object on page 9-33
- Type Property on page 11-164

FindFirst, FindLast, FindNext, and FindPrevious Methods

Applies To

OraDynaset Object on page 9-30

Description

Find the indicated rows in the dynaset that matches the `FindClause`. The `FindClause` can be any valid `WHERE` clause without the `WHERE`. If the current `FindClause` matches the last clause from the previous find operation, then the current `FindClause` is not parsed again.

These methods move the current row directly to a matched row without calling any advisories except when the matched row is reached. If a matching row cannot be found, the `NoMatch` property is set to `True`, and the current row remains the same.

Usage

```
oradynaset.FindFirst FindClause
oradynaset.FindLast FindClause
oradynaset.FindNext FindClause
oradynaset.FindPrevious FindClause
```

Remarks

The following types of expressions can be used in the `FindClause`:

- Simple queries, such as `"deptno = 20"`
- Queries involving complex expressions, such as `"sal + 100 > 1000"`.
- SQL function calls, such as `"UPPER(ename) = 'SCOTT' "` or `"NVL(comm, 0) = 0"`.
- Subqueries, such as `"deptno in (select deptno from dept)"`.

The SQL `LIKE` operator does not work in multiple byte languages. Table or synonym `DUAL` is required in the user's schema. Date values are retrieved and compared in Visual Basic format, which is the format specified in the Control Panel. Therefore, date comparisons fail if any other format such as the default Oracle format, DD-MON-YYYY is used.

The SQL function `TO_CHAR (date, fmt)` cannot be used because the first argument must be a date value in native Oracle format, and OO4O only handles `'string dates'`.

The SQL function `TO_DATE` converts a string to a date, but OO4O converts it back to a string in Visual Basic format, as previously described, and the comparison may still fail.

The `FindPrevious` and `FindLast` methods in a `NO_CACHE` dynaset do not work; `NoMatch` is set to `True`.

Note: To avoid raising an error, check for `EOF` or `BOF` before calling a `Find` method.

Examples

This example demonstrates the use of the `FindFirst`, `FindNext`, `FindPrevious` methods. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

  Dim OraSession As OraSession
  Dim OraDatabase As OraDatabase
  Dim OraDynaset As OraDynaset
  Dim OraFields As OraFields
  Dim FindClause As String

  Set OraSession = CreateObject("OracleInProcServer.XOraSession")
  Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "SCOTT/TIGER", 0&)
  Set OraDynaset = OraDatabase.CreateDynaset("select * from emp where empno" & _
                "">= 7654 and empno <= 7844 ", ORADYN_NO_BLANKSTRIP)

  Set OraFields = OraDynaset.Fields

  OraDynaset.MoveFirst

  'FindClause for job as MANAGER
  FindClause = "job LIKE '%GER'"

  OraDynaset.FindFirst FindClause

  'NoMatch property set to true , if no rows found
  If OraDynaset.NoMatch Then
    MsgBox "Couldn't find rows "
  else
    MsgBox OraFields("ename").Value  ' Should display BLAKE

    OraDynaset.FindNext FindClause
    MsgBox OraFields("ename").Value  ' Should display CLARK

    OraDynaset.FindPrevious FindClause
    MsgBox OraFields("ename").Value  ' Should display BLAKE

   endif

End Sub
```

See Also:

- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- NoMatch Property on page 11-110
- OraDatabase Object on page 9-28

Floor (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the floor, that is, lowest value, of an `OraNumber` object.

Usage

```
OraNumber.Floor
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

FlushBuffer (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Flushes, that is, empties, the content of the LOB to the database if LOB buffering has been enabled.

Usage

```
OraBlob.FlushBuffer
OraClob.FlushBuffer
```

See Also: EnableBuffering (OraLOB) Method on page 10-139

GetDatabaseFromPool Method

Applies To

OraSession Object on page 9-58

Description

Returns the next available `OraDatabase` object from the pool.

Usage

```
GetDatabaseFromPool(long waitTime)
```

Arguments

The arguments for the method are:

Arguments	Description
waitTime	The number of milliseconds this call waits for an object to be available, if the pool contains the maximum number of objects and all are used.

Remarks

To retrieve an `OraDatabase` object from the pool, the `GetDatabaseFromPool` method is called. This function returns a reference to an `OraDatabase` object. If the pool does not contain the maximum number of objects allowed, and all objects in the pool are used, then an additional `OraDatabase` object is created implicitly. In addition, if a pool item contains an `OraDatabase` object that has been timed out, then a new object is created and returned. The `OraDatabase` object obtained from the pool is then marked as in use and is returned to the pool when the object is no longer referenced by the application.

Exceptions are raised by this call if:

- The connection pool does not exist.
- The pool contains no objects.
- A time-out has occurred.

The `LastServerErr` property of the `OraSession` object contains the code for the specific cause of the exception.

See Also: CreateDatabasePool Method on page 10-83

GetChunk Method

Applies To

OraField Object on page 9-33

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)
```

Arguments

The arguments for the method are:

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

Remarks

The GetChunk method typically retrieves the specified bytes from the local cache. If data is not found in the cache, then the GetChunk method requests it from the database. Data from all fields (except the LONG or LONG RAW field) in the dynaset are retrieved and compared to the cached values for consistency. If any changes have occurred since the last fetch, then the GetChunk method stops the operation which causes an error and returns a Null string.

If a LONG or LONG RAW field is less than 65280 bytes, it is quicker to retrieve the data using the Value property than using the GetChunk method. You cannot use the GetChunk method on a LONG or LONG RAW field for which you have created an alias.

See "Migration from LONG RAW to LOB or BFILE" on page 5-5.

Examples

This example demonstrates the use of the GetChunk method to retrieve a LONG RAW column of a database and save it as 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 file name.

```
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").GetChunk(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
```

See Also:

- "Migration from LONG RAW to LOB or BFILE" on page 5-5
- AppendChunk Method on page 10-28
- FieldSize Method on page 10-150
- GetChunk Method on page 10-156
- Type Property on page 11-164
- Value Property on page 11-173
- OraField Object on page 9-33

GetChunkByte Method

Applies To

OraField Object on page 9-33

Description

Reads the data from the LONG or LONG RAW field into byte array and returns the size of data read.

Usage

```
Size_read = orafield.GetChunkByte(ByteArray, offset, numbytes)
```

Arguments

The arguments for the method are:

Arguments	Description
ByteArray	The first element of the ByteArray to hold the data.
offset	The number of bytes in the field to skip before copying data.
numbytes	The number of bytes to copy.

Remarks

When possible, the GetChunkByte method retrieves the specified bytes from the local cache. However, to conserve resources, some of the data might not be stored locally. In these cases, the GetChunkByte method 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 the GetChunkByte method stops the operation and an error occurs. In the case of an abort, the returned string is Null.

If a LONG or LONG RAW field is less than 65280 bytes in size, it is quicker to retrieve the data using the Value property than using the GetChunkByte method. You cannot use the GetChunkByte method on a LONG or LONG RAW field for which you have created an alias.

Examples

This example demonstrates the use of the GetChunkByte method to retrieve a LONG RAW column of a database and save it as 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 file name.

```
Sub GetChunkByteExample (FName As String)

'Declare various variables
Dim CurSize As Integer, ChunkSize  As Long
Dim I As Integer, FNum As Integer, CurChunk() As Byte

'Set the size of each chunk
```

```
ChunkSize = 10240
'Redim CurChunk Array
ReDim CurChunk(ChunkSize)

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
 CurSize = OraDynaset.Fields("type_longraw").GetChunkByte(CurChunk(0), I *
ChunkSize, ChunkSize)

If CurSize > 0 AND CurSize < ChunkSize Then
    ReDim CurChunk(CurSize)
    CurSize = OraDynaset.Fields("type_longraw").GetChunkByte(CurChunk(0), I *
ChunkSize, CurSize)
 End If
 Put #FNum, , CurChunk   'Write chunk to file.
 I = I + 1
Loop Until CurSize <= 0

'Close the file.
Close FNum

frmChunk.MousePointer = DEFAULT

End Sub
```

See Also: "Migration from LONG RAW to LOB or BFILE" on page 5-5 for additional information

GetChunkByteEx Method

Applies To

OraField Object on page 9-33

Description

Reads the data from a LONG or LONG RAW field into a Variant and returns the amount of data read.

Usage

```
amount_read = orafield.GetChunkByteEx(ByteArray, offset, numbytes)
```

Arguments

The arguments for the method are:

Arguments	Description
ByteArray	The name of the Variant ByteArray to hold the data.
offset	The number of bytes in the field to skip before copying data.
numbytes	The number of bytes to copy.

Remarks

When possible, the GetChunkByteEx method retrieves the specified bytes from the local cache. However, to conserve resources, some of the data might not be stored locally. In these cases, the GetChunkByteEx method 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 to the cached values for consistency. If any changes have occurred since the fetch of the original partial data, then the GetChunkByteEx method aborts the operation with an error.

Because the GetChunkByteEx method takes in a Variant as the first parameter, instead of the first element of the ByteArray as in the GetChunkByte method, only the GetChunkByteEx method can be used within an ASP/IIS environment.

If a LONG or LONG RAW field is less than 65280 bytes in size, it is quicker to retrieve the data using the Value property than using the GetChunkByteEx method.

See "Migration from LONG RAW to LOB or BFILE" on page 5-5.

Examples

Using the GetChunkByteEx Method to Retrieve a LONG RAW Example

This example demonstrates the use of the GetChunkByteEx method to retrieve a LONG RAW column of a database and save it as a file. This example expects a valid dynaset named OraDynaset representing a table with a column named type_longraw. Copy and paste this code into the definition section of a form. Call this procedure with a valid file name.

```
Sub GetChunkByteExExample (FName As String)
'Declare various variables
Dim bytesread As Integer, ChunkSize As Long ,
```

```
bytearr() as byte
Dim I As Integer, FNum As Integer, CurChunk
'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
  bytesread = OraDynaset.Fields("type_longraw").GetChunkByteEx(CurChunk,_
              I * ChunkSize, ChunkSize)
'redim byte array
redim bytearr(bytesread - 1)
bytearr = CurChunk
Put #FNum, , bytearr 'Write chunk to file.
I = I + 1
Loop Until bytesread < ChunkSize
'Close the file.
Close FNum
frmChunk.MousePointer = DEFAULT
End Sub
```

Using the GetChunkByteEx Method with Active Server Pages (ASP) Example

```
'This example is for use with ASP (Active Server Pages)
<%@ LANGUAGE = VBScript %>
<%Response.ContentType = "image/JPEG"%>
<%
Dim OraDatabase, Oradynaset
Dim Chunksize, BytesRead, CurChunkEx
'This assumes a pool of database connections have been created in the global.asa
Set OraDatabase = OraSession.getDatabaseFromPool(10)
'This assumes a table called "art_gallery" and
'displays JPEG images stored in the table
Set OraDynaset = OraDatabase.CreateDynaset("select art from art_gallery " & _
              "where artist = 'Picasso'", 0)

BytesRead = 0
'Reading in 32K chunks
ChunkSize= 32768
Do
  BytesRead = OraDynaset.Fields("picture").GetChunkByteEx(CurChunkEx, _
                                  i * ChunkSize, ChunkSize)
  if BytesRead > 0 then
     Response.BinaryWrite CurChunkEx
   end if
Loop Until BytesRead < ChunkSize
'Cleanup, remove all local references
Set OraDynaset = Nothing
Set Oradatabase = Nothing
%>
```

See Also: "Migration from LONG RAW to LOB or BFILE" on page 5-5

GetXML Method

Applies to

OraDynaset Object on page 9-30

Description

Generates an XML document based on the contents of the dynaset.

Usage

```
XMLstring = oradynaset.GetXML(startrow, maxrows)
```

Arguments

The arguments for the method are:

Arguments	Description
startrow	The row identifier indicating from which row to start (see `OraDynaset.RowPosition`). The default value of this argument is zero (the first row).
maxrows	The maximum number of rows to retrieve (if the end of the record set is reached; fewer rows may be returned). If this argument is omitted, then all rows are returned.

Remarks

This method returns a string containing the XML document.

The formatting of the output XML can be customized through the XML properties of the `OraDynaset` and `OraField` objects.

See Also:

- OraDynaset Object on page 9-30
- OraField Object on page 9-33
- RowPosition Property on page 11-132

GetXMLToFile Method

Applies To

OraDynaset Object on page 9-30

Description

Generates an XML document and writes it to a file.

Usage

```
oradynaset.GetXMLToFile (filename, startrow, maxrows)
```

Arguments

The arguments for the method are:

Arguments	Description
filename	The file name that the XML is written to. Existing files by the same name are overwritten.
startrow	The row identifier indicating from which row to start (see `OraDynaset.RowPosition`). The default value of this argument is `0` (the first row).
maxrows	The maximum number of rows to retrieve (if the end of the record set is reached; fewer rows may be returned). If this argument is omitted, then all rows are returned.

Remarks

There is no return value.

The formatting of the XML output can be customized through the XML properties of the `OraDynaset` and `OraField` objects.

See Also:

- OraDynaset Object on page 9-30
- OraField Object on page 9-33
- RowPosition Property on page 11-132

GetRows Method

Applies To

OraDynaset Object on page 9-30

Description

Retrieves multiple records of a dynaset object into `Variant` safe array.

Usage

```
Array =OraDynaset.GetRows(num_rows, start, fields )
```

Arguments

The arguments for the method are:

Arguments	Description
num_rows [optional]	An `Integer` representing the number of records to retrieve. Default value is the total number of rows in the dynaset.
start [optional]	An `Integer` representing the starting position of the dynaset from which the `GetRows` operation begins. Default value is the current position of the dynaset.
fields [optional]	A `Variant` representing a single field name or field position, or an array of field names or array of field position numbers. The `GetRows` method returns only the data in these fields.

Remarks

Use the `GetRows` method to copy records from a dynaset into a two-dimensional array. The first subscript identifies the field and the second identifies the row number. The `Array` variable is automatically dimensioned to the correct size when the `GetRows` method returns the data.

Calling the `GetRows` method does not change the current row position of the dynaset object.

Examples

The following example retrieves data using the `GetRows` method.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim row, col As Integer
Dim fields() As String

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

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

Set OraDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)
```

```
'The following line executes GetRows to get all records
data_array = OraDynaset.GetRows()

'Now display all the data in data_array
For row = 0 To UBound(data_array, 2)
    For col = 0 To UBound(data_array, 1)
        Debug.Print data_array(col, row)
    Next col
Next row

'The following lines execute GetRows to get the data from
'the ename and empno fields starting at 5

ReDim fields(2)

fields(0) = "EMPNO"
fields(1) = "ENAME"

'Execute GetRows
data_array = OraDynaset.GetRows(, 5, fields)

'Now display all the data in data_array
For row = 0 To UBound(data_array, 2)
    For col = 0 To UBound(data_array, 1)
        Debug.Print data_array(col, row)
    Next col
Next row
```

Get_Value Method

Applies To

OraParamArray Object on page 9-47

Description

Returns the value of a particular element of the array at the specified index.

Usage

```
OraParamArray.Get_Value(array, index)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *array*	A `String` representing the name of the array.
[in] *index*	An `Integer` representing the index value of the object.

Remarks

The `OraParamArray.Get_Value` method returns the value of the field as a `Variant`. The value of `data_value = oraparameter.Value` sets the contents of the parameter.

Note that fields of type `DATE` are returned in the default Visual Basic format as specified in the Control Panel, even though the default Oracle date format is "DD-MMM-YY".

The `Value` argument can be an Oracle Database 10*g* object, such as an `OraBLOB` object. For `Put_Value`, a copy of the object is made at that point in time, and `Get_Value` must be accessed to obtain a new object that refers to that index value. For example, if `iotype` is `ORATYPE_BOTH` and an `OraBLOB` object obtained from a dynaset is passed in as the input value, `Get_Value` needs to be called after the SQL code has been executed to obtain the newly updated output value of the `ParamaterArray` object.

Similar to a dynaset, the object obtained from the `ParamaterArray` `Get_Value` property refers to the latest value for that `ParamaterArray` index. The Visual Basic value `Null` can also be passed as a value. The Visual Basic value `EMPTY` can be used for `BLOB` and `CLOB` to indicate an empty LOB, and for `Object`, `VARRAY`, and nested table data types to indicate an object whose attributes are all `Null`.

This method is not available at design time and is read-only at run time.

When binding to `RAW` columns (`ServerType ORATYPE_RAW_BIN`), the value should be a byte array.

HypCos (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the hyperbolic cosine of an `OraNumber` object.

Usage

```
OraNumber.HypCos
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

HypSin (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the hyperbolic sine of an `OraNumber` object.

Usage

```
OraNumber.HypSin
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

HypTan (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the hyperbolic tangent of an `OraNumber` object.

Usage

```
OraNumber.HypTan
```

Remarks

The result of the operation is stored in an `OraNumber` object. There is no return value.

InitIterator Method

Applies To

OraCollection Object on page 9-19

Description

Initializes an iterator to scan a collection.

Usage

```
OraCollection.InitIterator
```

Remarks

This method initializes an iterator to point to the beginning of a collection. If this method is called for same Oracle Database 10*g* collection instance, then this method resets the iterator to point back to the beginning of the collection. The `OraCollection` object automatically reinitializes the iterator when the underlying collection changes due to a dynaset row navigation or a parameter `Refresh` method.

After you call the `InitIterator` method, you need to call the `IterNext` method or the first element in the collection repeats an extra time.

Examples

See "Example: OraCollection Iterator" on page 10-88.

See Also:

- IterNext Method on page 10-187
- IterPrev Method on page 10-188

IsEqual (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Checks if the `OraIntervalDS` object is equal to an argument.

Usage

```
isEqual = OraIntervalDSObj.IsEqual value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalDS` object is equal to the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If *value* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` object represents.

IsEqual (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Checks if the `OraIntervalYM` object is equal to an argument.

Usage

```
isEqual = OraIntervalYMObj.IsEqual value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalYM` object is equal to the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] YEARS-MONTHS.

If *value* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

IsEqual (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Checks if an `OraNumber` object is equal to an argument value.

Usage

```
bool  = OraNumber.IsEqual value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

Returns a Boolean value: The value is `True` if all values are equal; otherwise, it is `False`.

IsEqual (OraTimeStamp) Method

Applies To

OraTimeStamp Object on page 9-62

Description

Checks if the `OraTimeStamp` object is equal to an argument.

Usage

```
isEqual = OraTimeStampObj.IsEqual value format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStamp` to be compared.
[in] [optional] *format*	Specifies the `TIMESTAMP` format string to be used to interpret *value* when value is of type `String`. If *format* is not specified, the value is interpreted using the `Format` property of the current `OraTimeStamp` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStamp` object is equal to the argument; otherwise, it is `False`. The `IsEqual` method compares all the date-time values stored in the `OraTimeStamp` object.

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current `OraTimeStamp` object.

IsEqual (OraTimeStampTZ) Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Checks if the `OraTimeStampTZ` object is equal to an argument.

Usage

```
isEqual = OraTimeStampTZOb.IsEqual value, format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStampTZ` to be compared.
[in] [optional] *format*	Specifies the `TIMESTAMP WITH TIME ZONE` format string to be used to interpret *value* when *value* is type `String`. If *format* is not specified, *value* is interpreted using the `Format` property of the current `OraTimeStampTZ` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStampTZ` object is equal to the argument; otherwise, it is `False`. The `IsEqual` method only compares the Coordinated Universal Time (UTC) date-time values stored in the `OraTimeStampTZ` object; the time zone information is ignored.

> **Note:** UTC was formerly known as Greenwich Mean Time.)

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current session `OraTimeStampTZ` object.

If *value* is of `Date` type, the date-time value in `Date` is interpreted as the date-time value in the time zone of the session.

IsGreater (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Checks if the `OraIntervalDS` object is greater than an argument.

Usage

```
isGreater = OraIntervalDSObj.IsGreater value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalDS` object is greater than the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: Day [+/-] HH:MI:SSxFF.

If *value* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` object represents.

IsGreater (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Checks if the `OraIntervalYM` object is greater than an argument.

Usage

```
isGreater = OraIntervalYMObj.IsGreater value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalYM` object is greater than the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] YEARS-MONTHS.

If *value* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

IsGreater (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Checks if an `OraNumber` object is greater than an argument value.

Usage

```
bool  = OraNumber.IsGreater value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `OraNumber` object, or a numeric value.

Remarks

Returns a Boolean value: The value is `True` if the `OraNumber` object is greater than the argument; otherwise, it is `False`.

IsGreater (OraTimeStamp) Method

Applies To

OraTimeStamp Object on page 9-62

Description

Checks if the `OraTimeStamp` object is greater than an argument.

Usage

```
isGreater = OraTimeStampObj.IsGreater value format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStamp` to be compared.
[in] [optional] *format*	Specifies the `TIMESTAMP` format string to be used to interpret value when *value* is of type `String`. If *format* is not specified, the value is interpreted using the `Format` property of the current `OraTimeStamp` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStamp` object is greater than the argument; otherwise, it is `False`. The `IsGreater` method compares all the date-time values stored in the `OraTimeStamp` object.

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current `OraTimeStamp` object.

IsGreater (OraTimeStampTZ) Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Checks if the `OraTimeStampTZ object` is greater than an argument.

Usage

```
isGreater = OraTimeStampTZObj.IsGreater value, format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStampTZ` object to be compared.
[in] [optional] *format*	Specifies the `TIMESTAMP WITH TIME ZONE` format string to be used to interpret a value when *value* is type `String`. If *format* is not specified, *value* is interpreted using the `Format` property of the current `OraTimeStampTZ` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStampTZ` object is greater than the argument; otherwise, it is `False`. The `IsGreater` method only compares the UTC date-time values stored in the `OraTimeStampTZ` object; the time zone information is ignored.

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current `OraTimeStampTZ` object.

If *value* is of type `Date`, the date-time value in `Date` is interpreted as the date-time value in the time zone of the session.

IsLess (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Checks if the `OraIntervalDS` object is less than an argument.

Usage

```
isLess = OraIntervalDSObj.IsLess value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalDS` object is less than the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If *value* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS object` represents.

IsLess (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Checks if the `OraIntervalYM object` is less than an argument.

Usage

```
isLess  = OraIntervalYMObj.IsLess value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object to be compared.

Remarks

Returns a Boolean value: The value is `True` if the `OraIntervalYM` object is less than the argument; otherwise, it is `False`.

If *value* is a `Variant` of type `String`, it must be in the following format: [+/-] YEARS-MONTHS.

If *value* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

IsLess (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Checks if an `OraNumber` object is less than an argument value.

Usage

```
bool  = OraNumber.IsLess value
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `OraNumber` object, or a numeric value.

Remarks

Returns a Boolean value: The value is `True` if the OraNumber object is less than the argument; otherwise, it is `False`.

IsLess (OraTimeStamp) Method

Applies To

OraTimeStamp Object on page 9-62

Description

Checks if the `OraTimeStamp` object is less than an argument.

Usage

```
isLessr = OraTimeStampObj.IsLess value format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStamp`.
[in] [optional] *format*	Specifies the `TIMESTAMP` format string to be used to interpret value when value is of type `String`. If format is not specified, the value is interpreted using the `Format` property of the current `OraTimeStamp` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStamp` is less than the argument; otherwise, it is `False`. The `IsLess` method compares all the date-time values stored in the `OraTimeStamp` object.

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current `OraTimeStamp` object.

IsLess (OraTimeStampTZ) Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Checks if the `OraTimeSTampTZ` object is less than an argument.

Usage

```
isLess = OraTimeStampTZObj.IsLess value, format
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A `Variant` of type `String`, `Date`, or `OraTimeStampTZ`.
[[in] [optional] *format*	Specifies the `TIMESTAMP WITH TIME ZONE` format string to be used to interpret value when *value* is type `String`. If *format* is not specified, *value* is interpreted using the `Format` property of the current `OraTimeStampTZ` object.

Remarks

Returns a Boolean value: The value is `True` if the `OraTimeStampTZ` object is less than the argument; otherwise, it is `False`. `IsLess` only compares the UTC date-time values stored in the `OraTimeStampTZ` object; the time zone information is ignored.

If *value* is of type `String`, the string format must match the format specified in the *format* argument. If *format* is not specified, the string format must match the `Format` property of the current `OraTimeStampTZ` object.

If *value* is of type `Date`, the date-time value in `Date` is interpreted as the date-time value in the time zone of the session.

IterNext Method

Applies To

OraCollection Object on page 9-19

Description

Moves the iterator to point to the next element in the collection.

Usage

```
OraCollection.IterNext
```

Remarks

Using an iterator is faster than using an index when accessing collection elements.

If the iterator is pointing to the last element of the collection before to executing this function, then calling this method makes the `EOC` property return `True`. Also, the iterator is not changed. Check the `EOC` property when calling this method repetitively.

Call the `IterNext` method after the `InitIterator` method, or the first element in the collection is repeated an extra time.

Examples

See "Example: OraCollection Iterator" on page 10-88.

See Also:

- IterPrev Method on page 10-188
- InitIterator Method on page 10-171

IterPrev Method

Applies To

OraCollection Object on page 9-19

Description

Moves the iterator to point to the previous element in the collection.

Usage

```
OraCollection.IterPrev
```

Remarks

Using an iterator is faster than using an index when accessing collection elements.

If the iterator is pointing to the first element of the collection prior to executing this function, then calling this method makes the `BOC` property return `True`. Also, the iterator is not changed. Check the `BOC` property when calling this method repetitively.

Examples

See "Example: OraCollection Iterator" on page 10-88.

See Also:

- IterNext Method on page 10-187
- InitIterator Method on page 10-171

LastServerErrReset Method

Applies To

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

Clears the `LastServerErr` property to a zero value and sets the `LastServerErrText` property to `Null` for the specified object.

Usage

```
oradatabase.LastServerErrReset
orasession.LastServerErrReset
```

Remarks

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

See Also:

- OraDatabase Object on page 9-28
- OraSession Object on page 9-58
- LastServerErr Property on page 11-87
- LastServerErrText Property on page 11-90

Ln (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the natural logarithm (base *e*) of an `OraNumber` object.

Usage

```
OraNumber.Ln
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method raises an error if the `OraNumber` object is less than or equal to zero.

Log (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the logarithm of *operand* using the `OraNumber` object as the base.

Usage

```
OraNumber.Log operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method raises an error if the `OraNumber` object or *operand* is less than or equal to zero.

MatchPos (OraLOB/BFILE) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Returns the position of the *nth* occurrence of the pattern starting at the offset.

Usage

```
position = OraBlob.MatchPos pattern, offset, nth
position = OraClob.MatchPos pattern, offset, nth
position = OraBFile.MatchPos pattern, offset, nth
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *pattern*	A string for CLOB, or byte array for BLOB or BFILE that is searched for in the LOB.
[in] *Offset*	The starting position in the LOB or BFILE for the search.
[in] *nth*	The occurrence number.

Remarks

This call is currently implemented by executing a PL/SQL block that uses DBMS_LOB.INSTR().

Mod (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Gets the modulus from the division of the `OraNumber` object by *operand*.

Usage

```
OraNumber.Mod operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

If *operand* is equal to zero, an error is raised.

MonitorForFailover Method

Applies To

OraDatabase Object on page 9-28

Description

Registers the failover notification handler of the application.

Usage

```
OraDatabase.MonitorForFailover FOSink, FOCtx
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *FOSink*	An `IDispatch` interface implementing the `OnFailover` method which is notified in event of a failover.
[in] *FOCtx*	Context-specific information that the application wants passed into the `OnFailover` method in the event of a failover.

Remarks

To receive failover notifications, a notification handler must be registered with the `MonitorForFailover` method. The notification handler must be an automation object (a class module in Visual Basic) that implements the `OnFailover` method.

The syntax of the method is:

```
Public Function OnFailover(Ctx As Variant, fo_type As Variant,fo_event as Variant,
fo_OraDB as Variant)
```

Variants	Description
`[in] Ctx`	Passed into the `MonitorForFailover` method by the application. Context-sensitive information that the application wants passed in event of a failover.
`[in] fo_type`	`Failover` type. This is the type of failover that the client has requested. The values are: ■ `OO4O_FO_SESSION` indicates only session failover requested. ■ `OO4O_FO_SELECT` indicates select failover and session failover requested.

Variants	Description
[in] fo_event	Failover event. This indicates the state of the failover. It has several possible values: ■ OO4O_FO_BEGIN indicates that failover has detected a lost connection and failover is starting. ■ OO4O_FO_END indicates successful completion of a failover. ■ OO4O_FO_ABORT indicates that a failover was unsuccessful, and there is no option of retrying. ■ OO4O_FO_ERROR indicates that a failover was unsuccessful, and gives the application the opportunity to handle the error and retry the failover. The application can retry the failover, by programming the OnFailover method to return OO4O_FO_RETRY. ■ OO4O_FO_REAUTH indicates that a user handle has been reauthenticated. This applies to the situation where a client has multiple user sessions on a single database connection. During the initial failover, only the active user session is failed over. Other sessions are failed over when the application tries to use them. This is the value passed to the callback during these subsequent failovers.
[in] fo_OraDB	The OraDatabase object of the user session that is being failed over. Valid only when the fo_event variant is OO4O_FO_REAUTH.

Examples

Failover Notification Example

See Example: Failover Notification on page 4-25.

See Also: "Application Failover Notifications" on page 4-24

MonitorStart (OraAQ) Method

Applies To

OraAQ Object on page 9-3

Description

Starts a monitor thread for dequeuing the messages specified.

Usage

```
Q.MonitorStart NotificationHandler, CallbackCtx, MsgFilterVal,MsgFilter
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *NotificationHandler*	An IDispatch interface containing the callback method (NotifyMe) which should be notified of new messages.
[in] *CallbackCtx*	Context-specific information that the application wants to pass to the NotifyMe method. This is passed into the NotifyMe method whenever a new message satisfying the user criteria is dequeued.
[in] [optional] *MsgFilterVal*	A byte array containing a value for the message filter. Ignored if *MsgFilter* is ORAAQ_ANY.
[in] [optional] *MsgFilter*	An Integer constant specifying the selection criteria for messages. Possible values for *MsgFilter* are: ■ ORAAQ_ANY = 0 - Invokes the callback for any message that arrives on the queue. This is the default value. ■ ORAAQ_CONSUMER = 1 - Invokes the callback when the message intended for the consumer given in the MsgFilterValue is dequeued. ■ ORAAQ_MSGID = 2 - Invokes the callback when message with the identifier specified in *MsgFilterVal* is dequeued.

Remarks

NotifyMe is the callback method of the notification object. The syntax of the method is:

```
Public Sub NotifyMe (ByVal Ctx As Variant, ByVal Msgid As Variant)
```

Variants	Description
[in] Ctx	Value passed into the MonitorStart method by the application. Context-sensitive information that the application wants to pass in when messages are dequeued.
[in] Msgid	The message ID of the newly dequeued message. The Msgid variant is null when there is an error while monitoring.

By default, the message is passed into `NotifyMe` in `Remove` mode. The default dequeue options can be overridden by setting the properties of this instance (`OraAQ`).

The `MonitorStart` method returns `ORAAQ_SUCCESS` or `ORAAQ_FAIL`.

See Also: "Monitoring Messages" on page 4-21

MonitorStop (OraAQ) Method

Applies To

OraAQ Object on page 9-3

Description

Stops the monitor thread that was started earlier.

Usage

```
Q.MonitorStop
```

Remarks

Does nothing if a monitor is not running.

See Also: "Monitoring Messages" on page 4-21

MoveFirst, MoveLast, MoveNext, and MovePrevious Methods

Applies To

OraDynaset Object on page 9-30

Description

Change the cursor position to the first, last, next, or previous row within the specified dynaset. These move methods move the cursor to the next (previous, and so on) valid row, skipping rows that have been deleted.

Usage

```
oradynaset.MoveFirst
oradynaset.DbMoveFirst

oradynaset.MoveLast
oradynaset.DbMoveLast

oradynaset.MovePrevious
oradynaset.DbMovePrevious

oradynaset.MoveNext
oradynaset.DbMoveNext
```

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 the `MoveFirst` or `MoveLast` methods, respectively. You force the query to completion if you use the `MoveLast` method on a dynaset.

If you use the `MovePrevious` method and the first record is current, there is no current record and `BOF` is true. Using the `MovePrevious` method again causes an error, although `BOF` remains `True`. If you use the `MoveNext` method and the last record is current, there is no current record and `EOF` is true. Using the `MoveNext` method again causes an error, although `EOF` remains true. Note that when the dynaset is created with the `ORADYN_NO_MOVEFIRST` option, `BOF` and `EOF` are true whether the dynaset is empty or not.

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 the `Update` method is invoked automatically, although, it can 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, although no error occurs.

Data is fetched from the database, as necessary, so performing a `MoveFirst` operation followed by a `MoveNext` operation incrementally builds the mirrored (cached) local

set without requiring read-ahead of additional data. However, executing a `MoveLast` operation requires that the entire query be evaluated and stored locally.

When a dynaset is attached to a data control, these methods first notify the `Validate` event of the data control that record motion is about to occur. The `Validate` handler can 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.

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.CreateDynaset("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.MoveNext
 MsgBox "Employee #" & OraDynaset.Fields("empno").value & ", " & _
              OraDynaset.Fields("ename").value

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

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


End Sub
```

See Also:

- AddNew Method on page 10-21
- BOF Property on page 11-11
- EOF Property on page 11-56
- Edit Method on page 10-134
- EditMode Property on page 11-51
- RecordCount Property on page 11-128
- Update Method on page 10-257
- Validate Event on page 12-9

MovePreviousn, MoveNextn, MoveRel, and MoveTo Methods

Applies To

OraDynaset Object on page 9-30

Description

Change the cursor position to the specified row within the specified dynaset.

Usage

```
oradynaset.MovePreviousn offset
oradynaset.MoveNextn offset
oradynaset.MoveRel offset
oradynaset.MoveTo offset
```

MoveNextn Method

Moves offset records forward.

MovePreviousn Method

Moves offset records backward.

MoveRel Method

Moves offset records relative to the current row. A positive value, represented by a plus (+) sign, moves the cursor *down* the table, and a negative value moves the cursor up the table.

MoveTo Method

Moves directly to row number offset.

Remarks

`EOF` is set when the cursor moves beyond the end of a dynaset using `MoveNextn`, `MoveRel`, or `MoveTo` methods. `BOF` is set when the cursor moves beyond the start of a dynaset using `MovePreviousn`, `MoveRel`, or `MoveTo` methods. The `MoveNextn`, `MovePreviousn`, and `MoveTo` methods accept offset as a positive integer only. The `MoveRel` methods accepts offset as either a positive or a negative integer.

The `MoveTo` *`rownum`* always gets the same row unless the row has been deleted. If the requested row has been deleted, the `MoveTo` method moves to the next valid row. The `MoveNextn`, `MovePreviousn`, `MoveRel`, and `MoveTo` methods do not take into account deleted rows, so be cautious when using these methods based on relative positions of row numbers.

Data Type

`Long Integer`

Examples

This example demonstrates the use of the `MovePreviousn`, `MoveNextn`, `MoveRel`, and `MoveTo` methods. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Private Sub Form_Load()
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset
 Dim OraFields As OraFields

 Set OraSession = CreateObject("OracleInProcServer.XOraSession")
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "SCOTT/TIGER", 0&)
 Set OraDynaset = OraDatabase.CreateDynaset("select * from emp where empno" & _
           ">=7654 and empno <= 7844 ", ORADYN_NO LANKSTRIP)
 Set OraFields = OraDynaset.Fields

 'Move to 3rd record from the first record
 OraDynaset.MoveNextn 3  'Should set EOF to true
 MsgBox OraFields("ename").Value  ' Should be display SCOTT

 If OraDynaset.EOF = True Then
   MsgBox "End of the record reached"
 End If

 'Move back from the current record by the offset 2
 OraDynaset.MovePreviousn 2
 MsgBox OraFields("ename").Value  ' Should be display BLAKE

 If OraDynaset.BOF = True Then
   MsgBox "Start of the record reached"
 End If

 'Move relative in the forward direction
 OraDynaset.MoveRel 2
 MsgBox OraFields("ename").Value  ' Should be display SCOTT

 If OraDynaset.EOF = True Then
   MsgBox "End of the record reached"
 End If

 'Move relative in the backward direction
 OraDynaset.MoveRel -2
 MsgBox OraFields("ename").Value  ' Should be display BLAKE

 If OraDynaset.BOF = True Then
   MsgBox "Start of the record reached"
 End If

 'Move to the record position 4 in the current dynaset
  OraDynaset.MoveTo 4
  MsgBox OraFields("ename").Value  ' Should be display SCOTT

End Sub
```

Mul (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Multiplies the `OraIntervalDS` object by a multiplier.

Usage

```
OraIntervalDSObj.Mul multiplier
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *multiplier*	A `Variant` for type numeric value or an `OraNumber` object to be used as the multiplier.

Remarks

The result of the operation is stored in the `OraIntervalDS` object, overwriting any previous value. There is no return value.

Mul (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Multiplies the `OraIntervalYM` object by a multiplier.

Usage

```
OraIntervalYMObj.Mul multiplier
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *multiplier*	A `Variant` for type numeric value or an `OraNumber` object to be used as the multiplier.

Remarks

The result of the operation is stored in the `OraIntervalYM` object, overwriting any previous value. There is no return value.

Mul (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Multiplies the `OraNumber` object by *`operand`*.

Usage

```
OraNumber.Mul operand
```

Arguments

The arguments for the method are:

Arguments	Description
[`in`] *`operand`*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Neg (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Negates the `OraIntervalDS` object.

Usage

```
OraIntervalDSObj.Neg
```

Remarks

The result of the operation is stored in the `OraIntervalDS` object, overwriting any previous value. There is no return value.

Neg (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Negates the `OraIntervalYM` object.

Usage

```
OraIntervalYMObj.Neg
```

Remarks

The result of the operation is stored in the `OraIntervalYM` object, overwriting any previous value. There is no return value.

Neg (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Negates an `OraNumber` object.

Usage

```
OraNumber.Neg
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Open (OraServer) Method

Applies To

OraDatabase Object on page 9-28

OraServer Object on page 9-56

Description

Establishes a connection to an Oracle database.

Usage

```
OraServer.Open serverAlias
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *serverAlias*	A `String` containing the Network alias used for connecting to the database.

Remarks

If no arguments is supplied, this method attaches to a database that was detached previously.

See Also:

- BeginTrans Method on page 10-43
- Close Method on page 10-63
- CommitTrans Method on page 10-66
- CreateAQ Method on page 10-79
- CreateCustomDynaset Method on page 10-80
- CreateTempBLOB/CLOB Method on page 10-114
- CreateDynaset Method on page 10-85
- CreateOraObject (OraDatabase) Method on page 10-97
- Describe Method on page 10-124
- ExecuteSQL Method on page 10-144
- FetchOraRef Method on page 10-149
- LastServerErrReset Method on page 10-189
- MonitorForFailover Method on page 10-194
- RemoveFromPool Method on page 10-232

Open (OraBFILE) Method

Applies To

OraBFILE Object on page 9-9

Description

Opens a `BFILE`.

Usage

```
OraBfile.Open
```

Remarks

This method should be called before accessing the `BFILE` value.

OpenDatabase Method

Applies To

OraSession Object on page 9-58

OraServer Object on page 9-56

Description

Establishes a user session to the database. It creates a new `OraDatabase` object using the given database name, connection string, and specified options.

Usage

```
Set oradatabase = orasession.OpenDatabase(database_name,connect_string, options)
Set oradatabase = oraserver.OpenDatabase(connect_string, options)
```

Arguments

The arguments for the method are:

Arguments	Description
database_name	The Oracle Network specifier used when connecting the data control to a database.
connect_string	The user name 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 apply. The following table shows the possible modes, which can be combined by adding their respective values.

Constants

The following table lists constants and values for the options flag.

Constant	Value	Description
ORADB_DEFAULT	&H0&	Visual Basic Mode (Default): Field (column) values not explicitly set are set to `Null` when using the `AddNew` or `Edit` method. The `Null` values override any database column defaults. Wait on row locks when using `Edit ("SELECT...FOR UPDATE")`. Nonblocking SQL functionality is not enabled.
ORADB_ORAMODE	&H1&	Oracle Mode: Lets Oracle Database set the default field (column) values when using the `AddNew` method. The Oracle default column values are fetched again from database immediately after an insert or add operation. **Note:** If you use triggers, fetch the data again using the full Oracle Mode.

Constant	Value	Description
ORADB_NOWAIT	&H2&	Lock No-Wait Mode: Does not wait on row locks. When you use the Edit method to update a row that is locked by another user or process, Lock No-Wait mode results in an immediate return of an error code. **Note:** This option only applies to the OraDynaset object. It has no effect on OraSQLStmt objects or ExecuteSQL calls. It only raises an error in the case of a locked row.
ORADB_NO_REFETCH	&H4&	Oracle Mode (No Refetch): Performs like the Oracle Mode, but does not refetch data to the local cache. This boosts performance. **Note:** Use the No Refetch mode only when you intend to insert rows without editing them, because database column defaults cause inconsistencies between database data and the local cache. Attempting to edit after inserting in this mode causes a Data has been modified (4119) error.
ORADB_NONBLK	&H8&	Nonblocking Mode: Turns on Nonblocking mode on SQL statement execution. Nonblocking mode affects the SQL statements processed using the ExecuteSQL, CreateDynaset, or CreateSQL methods. **Note:** This feature has been deprecated.
ORADB_ENLIST_IN_MTS	&H10&	Enlist in MTS Mode: Determine whether the OraDatabase object enlists in the Microsoft Transaction Server (MTS) mode.
ORADB_ENLIST_FOR_CALLLBACK	&H20&	Enlist For Callbacks Mode: Turn on the event notification. This mode has to be enabled to receive Failover Notifications.

These values can be found in the oraconst.txt file. For creating a valid database alias, see the *Oracle Net Services Administrator's Guide*.

Examples of valid connect_string arguments include:

- "scott/tiger"
- "system/manager"
- "/"

Remarks

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 but does not perform any SQL actions.

One possible connection error that could be returned is:

```
ORA-28001 "the password has expired"
```

The user can change the password using the ChangePassword method.

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

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

End Sub
```

See Also:

- "Microsoft Transaction Server Support" on page 3-15
- ChangePassword (OraServer) Method on page 10-48
- "Application Failover Notifications" on page 4-24
- AddNew Method on page 10-21
- Edit Method on page 10-134
- ExecuteSQL Method on page 10-144
- ChangePassword (OraSession) Method on page 10-50
- OraConnection Object on page 9-27
- OraConnections Collection on page 9-66

OriginalItem Method

Applies To

OraFields Collection on page 9-67

Description

Returns the `OraField` object based on the original column name used in the `SELECT` statement in the dynaset. Not available at design time and read-only at run time.

Usage

```
set OraField = OraFields.OriginalItem(field_index)
set OraField = OraFields.OriginalItem(original_name)
```

Arguments

The arguments for the method are:

Arguments	Description
field_index	Field index of the original column name.
original_name	Original field name specified in the SQL statement.

Remarks

This is property is useful when a SQL statement contains `'schema.table.col'` as the `Name` of the field, and retrieves the field object specific to that original name.

Examples

The following example shows the use of the `OriginalItem` method. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset
 Dim OraFields As OraFields

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

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

 Set OraDynaset = OraDatabase.CreateDynaset("select scott.emp.deptno," & _
     "dept.deptno from scott.emp, scott.dept where dept.deptno = emp.deptno", 0&)

 'Get the Field collection object
 Set OraFields = OraDynaset.Fields

 'get the original field object. Returns "scott.emp.deptno"
```

```
 MsgBox OraField.OriginalName

 Set OraField = OraFields.OriginalItem(1)

 'Returns "dept.deptno"
 MsgBox OraField.OriginalName

End Sub
```

OriginalName

Applies To

OraField Object on page 9-33

Description

Returns the original column name used in the `SELECT` statement in the dynaset (as opposed to the name of the field as it appears on the server returned by the `Name` property). Not available at design time and read-only at run time.

Usage

```
field_name = Orafield.OriginalName
```

Remarks

The `orafield.OriginalName` method returns the name of the specified `OraField` object. This returns the `Original` column name specified in the SQL statement during dynaset creation. This property is useful when a SQL statement contains `'schema.table.col'` as the `Name` of the field. It enables duplicate column names to be referenced. (Duplicate column names can be avoided by using aliases in the SQL statement.)

Examples

The following example shows the use of the `OriginalName` property. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset
 Dim OraFields As OraFields

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

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

 Set OraDynaset = OraDatabase.CreateDynaset("select scott.emp.deptno," & _
    "dept.deptno from scott.emp, scott.dept where dept.deptno = emp.deptno", 0&)

 Set OraFields = OraDynaset.Fields

 'Returns "DEPTNO"
 MsgBox OraFields(0).Name

 'Returns "scott.emp.deptno"
 MsgBox OraFields(0).OriginalName

 'Returns "dept.deptno"
 MsgBox OraFields(1).OriginalName
```

```
End Sub
```

Power (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Raises the `OraNumber` object to the power of the operand.

Usage

```
OraNumber.Power operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Put_Value Method

Applies To

OraParamArray Object on page 9-47

Description

Inserts values into the table parameter.

Usage

```
OraParamArray.Put_Value(value, index)
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *value*	A Variant representing the value to insert.
[in] *index*	An Integer representing the index value of the object.

Remarks

This method should be used to insert a value before accessing a row in a table. A row does not contain a valid value until a row is assigned a value. Any reference to an unassigned row in the table raises an OLE Automation error.

The *value* argument can be an Oracle Database 10g object, such as an OraBLOB. For Put_Value, a copy of the object is made at that point in time, and Get_Value must be accessed to obtain a new object that refers to that index value. For example, if iotype is ORATYPE_BOTH and an OraBLOB obtained from a dynaset is passed in as the input value, Get_Value needs to be called after the SQL has been executed to obtain the newly updated output value of the ParamaterArray.

Similar to a dynaset, the object obtained from ParamaterArray Get_Value method always refers to the latest value for that ParamaterArray index. The Visual Basic value Null can also be passed as a value. The Visual Basic value EMPTY can be used for BLOB and CLOB to indicate an empty LOB, and for OBJECT, VARRAY and NESTED TABLE to indicate an object whose attributes are all Null.

When binding to RAW columns (ServerType ORATYPE_RAW_BIN) value should be a byte array.

Read (OraLOB/BFILE) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Reads into a buffer a specified portion of a BLOB, CLOB, or BFILE value. Returns the total amount of data read.

Usage

```
amount_read = OraBlob.Read buffer, chunksize
amount_read = OraClob.Read buffer, chunksize
amount_read = OraBfile.Read buffer, chunksize
```

Arguments

The arguments for the method are:

Arguments	Description
[out] *buffer*	Variant of type character array for OraCLOB, Variant of type byte array for OraBLOB, or OraBFILE from which the piece is read.
[in] [optional] *chunksize*	An Integer specifying the amount to be read. Default value is the size of the LOB. In bytes for OraBLOB or OraBFILE; characters for OraCLOB.
[out] *amount_read*	An Integer representing the total amount of data read. In bytes for OraBLOB or OraBFILE; characters for OraCLOB.

Remarks

Reads the LOB or BFILE data from the offset specified by the Offset property. For multiple piece read operation, the PollingAmount property must be set to the value of the total amount of data to be read, and the Status property must be checked for the success of each piece operation.

> **Note:** When reading a portion of a LOB, it is recommended that you set the PollingAmount property, rather than using the *chunksize* parameter. This avoids the possibility of raising an error if the entire LOB is not read before to executing another LOB method.

Examples

Be sure that you have installed the OraLOB Schema Objects as described in "Schema Objects Used in LOB Data Type Examples" on page A-3.

Example: Multiple-Piece Read of a LOB

```
Dim OraSession As OraSession
```

```
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim PartImage As OraBlob
Dim chunksize As Long
Dim AmountRead As Long
Dim buffer As Variant
Dim buf As String

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

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

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

'Get OraBlob from OraDynaset
Set PartImage = OraDynaset.Fields("part_image").Value

'Set Offset and PollingAmount property for piecewise Read operation
PartImage.offset = 1
PartImage.PollingAmount = PartImage.Size
chunksize = 50000

'Get a free file number
FNum = FreeFile

'Open the file
Open "image.dat" For Binary As #FNum

'Do the first read on PartImage, buffer must be a variant
AmountRead = PartImage.Read(buffer, chunksize)

'put will not allow Variant type
buf = buffer
Put #FNum, , buf

' Check for the Status property for polling read operation
While PartImage.Status = ORALOB_NEED_DATA
    AmountRead = PartImage.Read(buffer, chunksize)
    buf = buffer
    Put #FNum, , buf
Wend

Close FNum
```

Example: Single-Piece Read of a LOB

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim PartDesc As OraClob
Dim AmountRead As Long
Dim buffer As Variant
Dim buf As String

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

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

'Add PartDesc as an Output parameter and set its initial value.
OraDatabase.Parameters.Add "PartDesc", Null, ORAPARM_OUTPUT
OraDatabase.Parameters("PartDesc").ServerType = ORATYPE_CLOB

'Execute the statement returning 'PartDesc'
OraDatabase.ExecuteSQL ("BEGIN select part_desc into :PARTDESC from" & _
             "part where part_id = 1 for update NOWAIT; END;")

'Get 'PartDesc' from Parameters collection
Set PartDesc = OraDatabase.Parameters("PartDesc").Value

'Get a free file number
FNum = FreeFile

'Open the file.

Open "Desc.Dat" For Binary As #FNum

'Read entire CLOB value, buffer must be a Variant
AmountRead = PartDesc.Read(buffer)

'put will not allow Variant type
buf = buffer
Put #FNum, , buf

Close FNum
```

See Also:

- Offset (OraLOB/BFILE) Property on page 11-112
- PollingAmount Property on page 11-125
- Status (OraLOB/BFILE) Property on page 11-154

ReadChunk Method

Applies To

OraField Object on page 9-33

Description

Returns a `String` containing the bytes of all or a portion of a `LONG` or `LONG RAW` field.

Usage

```
data_string = orafield.ReadChunk(offset, numbytes, bytesread)
```

Arguments

The arguments for the method are:

Arguments	Description
offset	The number of bytes in the field to skip before copying data.
numbytes	The number of bytes to copy.
bytesread	The number of bytes read.

Remarks

The `ReadChunk` method behaves like the `GetChunk` method, but it returns the actual number of bytes read in the *bytesread* argument.

See Also:

- "Migration from LONG RAW to LOB or BFILE" on page 5-5
- GetChunk Method on page 10-156

Refresh Method

Applies To

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

Description

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

Forces an immediate update of the dynaset by reexecuting the SQL statement in the SQL statement object.

Usage

```
oradynaset.Refresh
oradynaset.DbRefresh
orasqlstmt.Refresh
orasqlstmt.DbRefresh
```

Remarks

This method cancels all edit operations (`Edit` and `AddNew` methods), 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 refreshing with a data control. This method also lets you execute a modified SQL statement without creating a new dynaset or `OraSQLStmt` object.

The preferred refresh methods when changing parameter values are `oradynaset.Refresh` or `orasqlstmt.Refresh`, because required database operations are minimized (SQL parsing, binding, and so on). This can improve performance when only parameter values have changed.

If you call the `Refresh` method after assigning an invalid SQL statement to the SQL property of a dynaset or SQL statement object, these objects remain valid. However, a dynaset in this state does not permit any row or field operations. Bound controls also 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.

You can regain the normal dynaset and SQL statement operations by refreshing the object with a valid SQL statement. The `Refresh` method treats `Null` or empty SQL statements as invalid.

Examples

Refresh Method Example (OraDynaset)

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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

 'Create the OraDynaset Object.
 Set OraDynaset =OraDatabase.CreateDynaset("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.Refresh

 '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
```

Refresh Method Example (OraSQLStmt)

This example demonstrates the use of parameters, the `Refresh` method, and the SQL property for the . object. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraSqlStmt As OraSQLStmt
```

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

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

OraDatabase.Parameters.Add "EMPNO", 7369, 1
OraDatabase.Parameters("EMPNO").ServerType = 2  'ORATYPE_NUMBER
OraDatabase.Parameters.Add "ENAME", 0, 2
OraDatabase.Parameters("ENAME").ServerType = 1  'ORATYPE_VARCHAR2

Set OraSqlStmt = OraDatabase.CreateSQL("Begin Employee.GetEmpName (:EMPNO," & _
             ":ENAME); end;", 0&)

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

'Should display SMITH
MsgBox OraDatabase.Parameters("ENAME").Value

'Change the value of the empno parameter.
OraDatabase.Parameters("EMPNO").Value = 7499

'Refresh the dynaset.
OraSqlStmt.Refresh

'Should display ALLEN
MsgBox OraDatabase.Parameters("ENAME").Value

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

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

End Sub
```

See Also:

- AddNew Method on page 10-21
- Connect Property on page 11-23
- CreateDynaset Method on page 10-85
- DatabaseName Property on page 11-37
- Edit Method on page 10-134
- OraConnection Object on page 9-27
- OraDynaset Object on page 9-30
- OraSession Object on page 9-58
- SQL Property on page 11-150
- RecordSource Property on page 14-31

Refresh (OraRef) Method

Applies To

OraRef Object on page 9-52

Description

Refreshes the referenceable object from the most current database snapshot.

Usage

```
OraRef.Refresh
```

Register Method

Applies To

OraSubscription Object on page 9-61

Description

Activates the subscription.

Usage

```
orasubscription.Register
```

Remarks

When the specified database event is fired, the `NotifyDBevents` method of the `dbevent` handler that was passed in while creating this subscription is invoked.

Examples

See "Example: Registering an Application for Notification of Database Events" on page 10-15 for a complete example.

See Also:

- "Database Events" on page 4-22
- OraSubscription Object on page 9-61
- OraSubscriptions Collection on page 9-70

Remove Method

Applies To

OraParameters Collection on page 9-68

Description

Removes a parameter from the `OraParameters` collection.

Usage

```
oraparameters.Remove(member_name)
```

Arguments

The arguments for the method are:

Arguments	Description
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 the `AutoBindDisable` and `AutoBindEnable` methods.

For an `OraParameter` of type `ORATYPE_CURSOR`, this method destroys the dynaset object associated with the cursor, and clears the local cache temporary files.

Examples

See "Example: ExecuteSQL" on page 10-145.

See Also:

- Add Method on page 10-8
- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41
- OraDatabase Object on page 9-28
- OraParameter Object on page 9-50
- OraParameters Collection on page 9-68

Remove (OraSubscriptions Collection) Method

Applies To

OraSubscriptions Collection on page 9-70

Description

Removes a subscription from the `OraSubscriptions` collection.

Usage

```
orasubscriptions.Remove(member)
```

Arguments

The arguments for the method are:

Arguments	Description
member	A `Variant` specifying an integer subscript from `0` to `Count`, or the subscription name.

Remarks

This method unregisters (removes) the subscription if it is active, and destroys the subscription associated with it.

See Also:

- Add (OraSubscriptions Collection) Method on page 10-14
- OraSubscription Object on page 9-61
- OraSubscriptions Collection on page 9-70

RemoveFromPool Method

Applies To

OraDatabase Object on page 9-28

Description

Removes the `OraDatabase` object from the pool.

Usage

```
OraDatabase.RemoveFromPool
```

Remarks

This method applies only to those `OraDatabase` objects that are retrieved from the pool using the `GetDatabaseFromPool` method.

No exceptions or errors are raised if the `OraDatabase` object is not a member the pool.

This method is useful for removing `OraDatabase` objects from the pool whose connections are no longer valid.

See Also:

- CreateDatabasePool Method on page 10-83
- DestroyDatabasePool Method on page 10-128
- GetDatabaseFromPool Method on page 10-155

ResetTrans Method

Applies To

OraConnection Object on page 9-27

OraSession Object on page 9-58

Description

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

Usage

```
oraconnection.ResetTrans
orasession.ResetTrans
```

Remarks

This method does not generate events or produce errors. Because the `ResetTrans` method does not generate events, you cannot cancel the `ResetTrans` method in a `Validate` event, as you can with a rollback or commit operation.

Note: If an `OraDatabase` object has been enlisted with Microsoft Transaction Server (MTS) and is part of a global MTS transaction, this method has no effect.

Examples

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

```
Sub Form_Load ()

'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

 'Start Transaction processing.
 OraDynaset.Session.BeginTrans

 'Traverse until EOF is reached, setting each employee's salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.Edit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.Update
   OraDynaset.MoveNext
 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.ResetTrans
 MsgBox "Salary changes rolled back."

End Sub
```

See Also:

- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- OraSession Object on page 9-58
- Rollback Method on page 10-235
- Microsoft Transaction Server Support on page 3-15
- Validate Event on page 12-9

Rollback Method

Applies To

OraConnection Object on page 9-27

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

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

Usage

```
oraconnection.Rollback
orasession.Rollback
oradatabase.Rollback
```

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 advised when the rollback succeeds.

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

- `OraConnection` and `OraDatabase`:

  The `Rollback` method rolls back all pending transactions within the specified connection. This method has no effect if a transaction has not begun. When a session-wide transaction is in progress, you can use this call to prematurely roll back the transactions for the specified connection.

- `OraSession`:

  The `Rollback` method rolls back all pending transactions within the specified session. The `Rollback` method is valid only when a transaction has been started. If a transaction has not been started, the use of the `Rollback` method results in an error.

> **Note:** If an `OraDatabase` object has been enlisted with Microsoft Transaction Server (MTS) and is part of a global MTS transaction, this method has no effect.

Examples

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

```
Sub Form_Load ()

'Declare variables
 Dim OraSession As OraSession
```

```
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

 'Start Transaction processing.
 OraDynaset.Session.BeginTrans

 'Traverse until EOF is reached, setting each employee's salary to zero.
 Do Until OraDynaset.EOF
   OraDynaset.Edit
   OraDynaset.Fields("sal").value = 0
   OraDynaset.Update
   OraDynaset.MoveNext
 Loop
 MsgBox "All salaries set to ZERO."

 'Currently, the changes have NOT been committed to the database.
 'End Transaction processing.
 OraDynaset.Session.Rollback
 MsgBox "Salary changes rolled back."

End Sub
```

See Also:

- AutoCommit Property on page 11-9
- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- OraSession Object on page 9-58
- OraConnection Object on page 9-27
- ResetTrans Method on page 10-233
- "Microsoft Transaction Server Support" on page 3-15
- Validate Event on page 12-9

Round (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Rounds the `OraNumber` object to the specified decimal place.

Usage

```
OraNumber.Power decplaces
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *decplaces*	An `Integer` specifying the number of digits to the right of the decimal point from which to round. Negative values are allowed and signify digits to the left of the decimal point.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

SetPi (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Sets an `OraNumber` object to Pi.

Usage

```
OraNumber.SetPi
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Sin (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the sine of an `OraNumber` object given in radians.

Usage

```
OraNumber.Sin
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Sqrt (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the square root of an `OraNumber` object.

Usage

```
OraNumber.Sqrt
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

This method returns an error if the `OraNumber` object is less than zero.

Sub (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Subtracts an argument from the `OraIntervalDS` object.

Usage

```
OraIntervalDSObj.Sub operand
```

Arguments

The arguments for the method are:

Arguments	Description
[`in`] *`operand`*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS`, object to be subtracted.

Remarks

The result of the operation is stored in the `OraIntervalDS` object, overwriting any previous value. There is no return value.

If *`operand`* is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If *`operand`* is a numeric value, the value provided should represent the total number of days that the constructed `OraIntervalDS` object represents.

Sub (OraIntervalYM) Method

Applies To

OraIntervalYM Object on page 9-37

Description

Subtracts an argument from the `OraIntervalYM` object.

Usage

```
OraIntervalYMObj.Sub operand
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *operand*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object to be subtracted.

Remarks

The result of the operation is stored in the `OraIntervalYM` object, overwriting any previous value. There is no return value.

If *operand* is a `Variant` of type `String`, it must be in the following format: [+/-] YEARS-MONTHS.

If *operand* is a numeric value, the value provided should represent the total number of years that the constructed `OraIntervalYM` object represents.

Sub (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Subtracts a numeric argument from the `OraNumber` object.

Usage

```
OraNumber.Sub operand
```

Arguments

The arguments for the method are:

Arguments	Description
[`in`] *operand*	A `Variant` of type `String`, type `OraNumber`, or a numeric value.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Tan (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Calculates the tangent of an `OraNumber` object given in radians.

Usage

```
OraNumber.Tan
```

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

ToDate Method

Applies To

OraTimeStamp Object on page 9-62

OraTimeStampTZ Object on page 9-64

Description

Returns a copy of the `Date` type from an `OraTimeStamp` or `OraTimeStampTZ` object.

Usage

```
Set date = OraTimeStampObj.ToDate
Set date = OraTimeStampTZObj.ToDate
```

Remarks

This method returns the datetime values in the `Date` data type. As a result, the date-time values can be adjusted if they fall outside the range allowed by a VB date.

- For an `OraTimeStamp` object:

  Returns a new `Date` object with the same date-time values as the current `OraTimeStamp` object, but the nanosecond portion is truncated.

- For an `OraTimeStampTZ` object:

  Returns a new `Date` object with the same date-time values as the current `OraTimeStampTZ` object, but the nanosecond portion and time zone portion are truncated.

Examples

Using the OraTimeStamp Object

```
Dim OraTimeStamp As OraTimeStamp

...
'Create OraTimeStamp using a string
Set OraTimeStamp = OraSession.CreateOraTimeStamp("1999-APR-29 12:10:23.444 AM", _
       "YYYY-MON-DD HH:MI:SS.FF AM")

' returns a Date type with date value set to "1999-APR-29 12:10:23 AM"
' note that the fractional part is dropped
Set date = OraTimeStamp.ToDate
```

Using the OraTimeStampTZ Object

```
Dim OraTimeStampTZ As OraTimeStampTZ

...
'Create OraTimeStampTZ using a string
Set OraTimeStampTZ = OraSession.CreateOraTimeStampTZ("2000-12-28" & _
        "12:10:23.444 -07:00", "YYYY-MM-DD HH:MI:SS.FF TZH:TZM")

'returns a Date type with date value set to "2000-12-28 12:10:23"
```

```
'note that Time Zone and nanosecond portions are dropped
Set date = OraTimeStampTZ.ToDate
```

ToOraNumber (OraIntervalDS) Method

Applies To

OraIntervalDS Object on page 9-35

Description

Returns an `OraNumber` object containing a value that represents the total number of days that the `OraIntervalDS` object specifies.

Usage

```
Set OraNumberObj = OraIntervalDSObj.ToOraNumber
```

ToOraTimeStamp Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns a copy of the `OraTimeStamp` object that has the date-time value in the specified time zone of the current `OraTimeStampTZ` object.

Returns a copy of the `OraTimeStamp` object from an `OraTimeStampTZ` object.

Usage

```
Set OraTimeStampObj = OraTimeStampTZObj.ToOraTimeStamp
```

Remarks

Returns a new `OraTimeStamp` object that has the date-time values in the specified time zone of the current `OraTimeStampTZ` object.

Examples

```
Dim OraTimeStampTZ As OraTimeStampTZ

...
'Create OraTimeStampTZ using a string
Set OraTimeStampTZ = OraSession.CreateOraTimeStampTZ("2000-12-28" & _
       "12:10:23.444 -07:00", "YYYY-MM-DD HH:MI:SS.FF TZH:TZM")

'returns a new OraTimeStamp object with date value equal to
' "2000-12-28 12:10:23.444"
'note that Time Zone portion is dropped
Set OraTimeStamp = OraTimeStampTZ.ToOraTimeStamp
```

ToOraTimeStampLTZ Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns a copy of the `OraTimeStamp` object that has the date-time value normalized to the session time zone of the current `OraTimeStampTZ` object.

Usage

```
Set OraTimeStampObj = OraTimeStampTZObj.ToOraTimeStampLTZ
```

Remarks

Returns a new `OraTimeStamp` object that has the date-time values normalized to the session time zone of the current `OraTimeStampTZ` object.

Examples

```
Dim OraTimeStampTZ As OraTimeStampTZ

...
'Create OraTimeStampTZ using a string
Set OraTimeStampTZ = OraSession.CreateOraTimeStampTZ("2003-APR-29" & _
        "12:00:00 -07:00", "YYYY-MON-DD HH:MI:SS TZH:TZM")

'Assuming that the Session Time Zone is "-08:00"
'returns a new OraTimeStamp object with date value normalized to
'session Time Zone, "2003-APR-29 11:00:00"

Set OraTimeStamp = OraTimeStampTZ.ToOraTimeStampLTZ
...
```

ToOraTimeStampTZ Method

Applies To

OraTimeStamp Object on page 9-62

Description

Returns a copy of the `OraTimeStampTZ` object from an `OraTimeStamp` object.

Usage

```
Set OraTimeStampTZObj = OraTimeStampObj.ToOraTimeStampTZ
```

Remarks

Returns a new `OraTimeStampTZ` object with the same date-time values as the current `OraTimeStamp` object. The time zone information in the returned `OraTimeStampTZ` object is set to the session time zone.

Examples

```
Dim OraTimeStamp As OraTimeStamp

...
'Create OraTimeStamp using a string
Set OraTimeStamp = OraSession.CreateOraTimeStamp("1999-APR-29" & _
         "12:10:23.444 AM", "YYYY-MON-DD HH:MI:SS.FF AM")

' assuming that the session Time Zone is "-07:00" returns a new
' OraTimeStampTZ object with date value equal to "1999-APR-29 12:10:23 -07:00"
Set OraTimeStampTZ = OraTimeStamp.ToOraTimeStampTZ
```

ToUniversalTime Method

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns a copy of the `OraTimeStampTZ` object that has the date-time value normalized to Coordinated Universal Time (UTC) of the current `OraTimeStampTZ` object.

Usage

```
Set OraTimeStampTZObj1 = OraTimeStampTZObj.ToUniversalTime
```

Remarks

Returns a new `OraTimeStampTZ` object that has the date-time values normalized to the UTC of the current `OraTimeStampTZ` object.

> **Note:** UTC was formerly known as Greenwich Mean Time.

Examples

```
Dim OraTimeStampTZ As OraTimeStampTZ
Dim OraTimeStampTZ_UTC As OraTimeStampTZ
...
'Create OraTimeStampTZ using a string
Set OraTimeStampTZ = OraSession.CreateOraTimeStampTZ("2003-APR-29 " & _
        "12:00:00 -07:00", "YYYY-MON-DD HH:MI:SS TZH:TZM")

'returns a new OraTimeStampTZ object with date value normalized to
'UTC time, "2003-APR-29 19:00:00 00:00"
Set OraTimeStampTZ_UTC = OraTimeStampTZ.ToUniversalTime
...
```

Trim (OraCollection) Method

Applies To

OraCollection Object on page 9-19

Description

Trims a given number of elements from the end of the collection.

Usage

```
OraCollection.Trim size
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *size*	An `Integer` specifying the number of elements to trim.

Remarks

The elements are removed from the end of the collection. An error is returned if the size is greater than the current size of the collection.

Examples

The following example illustrates the `Trim` method. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in OraCollection Examples" on page A-3.

Example: Trim Method for the OraCollection Object

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim EnameList as OraCollection

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

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

'create a dynaset object from department
set OraDynaset = OraDatabase.CreateDynaset("select * from department", 0&)

'retrieve a Enames column from Department.
'Here Value property of OraField object returns EnameList OraCollection
set EnameList = OraDynaset.Fields("Enames").Value

'display the size of the collection
msgbox  EnameList.Size
```

```
'Trim the EnameList collection by one. Before that row level
 'lock should be obtained

OraDynaset.Edit
EnameList.Trim 1
OraDynaset.Update

'display the new size of the collection
msgbox EnameList.Size
```

Trim (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Trims or truncates the LOB value to shorter length.

Usage

```
OraBlob.Trim NewLen
OraClob.Trim NewLen
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *NewLen*	An `Integer` specifying the new length of the LOB value; must be less than or equal to the current length.

Remarks

Either a row-level lock or object-level lock should be obtained before calling this method.

> **Note:** When manipulating LOBs using LOB methods, such as the `Write` and `CopyFromFile`, the LOB object is not automatically trimmed if the length of the new data is shorter than the old data. Use the `Trim` (OraLOB) method to shrink the LOB object to the size of the new data.

Trunc (OraNumber) Method

Applies To

OraNumber Object on page 9-41

Description

Truncates an Oracle number at a specified decimal place.

Usage

```
OraNumber.Trunc decplaces
```

Arguments

The arguments for the method are:

Arguments	Description
[in] *decplaces*	An `Integer` specifying the number of digits to the right of the decimal point from which to truncate. Negative values are allowed and signify digits to the left of the decimal point.

Remarks

The result of the operation is stored in the `OraNumber` object. There is no return value.

Unregister Method

Applies To

OraSubscription Object on page 9-61

Description

Unregisters this subscription, which turns off notifications on the specific database event.

Usage

```
orasubscription.UnRegister
```

Remarks

Unregistering a subscription ensures that the user does not receive notifications related to that subscription or database event in the future. If the user wants to resume notification, then the only option is to re-register the subscription.

Examples

Registering an Application for Notification of Database Events Example

See "Example: Registering an Application for Notification of Database Events" on page 10-15.

See Also:

- "Database Events" on page 4-22
- OraSubscription Object on page 9-61
- OraSubscriptions Collection on page 9-70

Update Method

Applies To

OraDynaset Object on page 9-30

Description

Saves the copy buffer to the specified dynaset.

Usage

```
oradynaset.Update
oradynaset.DbUpdate
```

Remarks

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

Once the `Update` method is called on a given row in a dynaset in a global transaction (that is, a `BeginTrans` operation is issued), locks remain on the selected rows until a `CommitTrans` or `Rollback` method is called.

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` method that created the `OraDatabase` object of this dynaset.

If this dynaset is attached to a data control, then the `Validate` event of the data control 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.

Examples

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

```
Sub Form_Load ()

'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

'Begin an AddNew.
OraDynaset.AddNew
```

```
'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.Update

End Sub
```

See Also:

- AddNew Method on page 10-21
- Edit Method on page 10-134
- OpenDatabase Method on page 10-212
- OraDatabase Object on page 9-28
- Validate Event on page 12-9

Update (OraRef) Method

Applies To

OraRef Object on page 9-52

Description

Flushes the modified referenceable object to the database.

Usage

```
OraRef.Update
```

Remarks

The `Update` method completes the `Edit` operation and commits the changes to the database unless a `BeginTrans` operation is pending for the session.

Examples

The following example updates the attributes of the `PERSON` referenceable object in the database. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

Updating Attribute Values: Dynaset Example

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)

'retrieve a aperson column from customers. Here Value property of OraField
'object returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'locks the Person object in the server for modifying its attributes
Person.Edit
  Person.Name = "Eric"
  Person.Age = 35

'Update method flushes the modified referenceable object in the server
Person.Update
```

Updating Attribute Values: Parameter Example

```
Dim OraSession as OraSession
```

```
Dim OraDatabase as OraDatabase
Dim Person  as OraRef

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

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

'create an  OraParameter object represent Address object bind Variable
OraDatabase.Parameters.Add "PERSON", Null, ORAPARM_OUTPUT, ORATYPE_REF,"PERSON"

'execute the sql statement which selects person from the customers table
OraDatabase.ExecuteSQL ("BEGIN select aperson into :PERSON from customers" & _
                  "where account = 10; END;")

'get the Person object from OraParameter
set Person = OraDatabase.Parameters("PERSON").Value

'locks the Person object in the server for modifying its attributes
Person.Edit
  Person.Name = "Eric"
  Person.Age = 35

'Update method flushes the modified referenceable object in the server
Person.Update
```

Write (OraLOB) Method

Applies To

OraBLOB, OraCLOB Objects on page 9-11

Description

Writes a buffer into the BLOB or CLOB value of this object and returns the total amount of the data written.

Usage

```
amount_written = OraBlob.Write buffer, chunksize, piece
amount_written = OraClob.Write buffer, chunksize, piece
```

Arguments

The arguments for the method are:

Arguments	Description
in] *buffer*	The character array for an OraCLOB object or byte array for the OraBLOB object from which the piece is written.
[in] [optional] *chunksize*	An Integer specifying the length of the buffer, in characters for an OraCLOB object and bytes for an OraBLOB or OraBFILE object. Default value is the size of the buffer argument.
[in] [optional] *piece*	An Integer specifying which piece of the buffer is being written. Possible values include: ■ ORALOB_ONE_PIECE - Buffer is written in a single piece. This is the default. ■ ORALOB_FIRST_PIECE - Buffer represents the piece of LOB data to be written. ■ ORALOB_NEXT_PIECE - Buffer represents the next piece of LOB data to be written. ■ ORALOB_LAST_PIECE - Buffer represents the last piece of LOB data to be written.
[out] *amount_written*	An Integer representing the amount written, in characters for an OraCLOB object and bytes for an OraBLOB or OraBFILE object.

Remarks

Obtain either a row-level lock or object-level lock before calling the Write method. This method writes the BLOB or CLOB data from the offset specified by the Offset property. For a multiple-piece write operation, the PollingAmount property can be set to the value of the total amount of data to be written, and the Status property must be checked for the success of each piece operation. If the total amount is not known, then the PollingAmount property can be set to 0 and polling still occurs as long as the piece type is not OraLob_piece.

For the last piece, set the piece argument to ORALOB_LAST_PIECE. You must write the polling amount in bytes or characters. It is not possible to terminate the Write operation early if the PollingAmount property is not zero.

When the OraLOB `Pollingamount = 0` but the piece type on OraLOB `Write` is not `ORALOB_ONE_PIECE`, polling still occurs. Polling completes when `ORALOB_LAST_PIECE` is sent as an argument to a call to the `Write` method. This is useful when calling the `OraCLOB.Write` method in a variable-width character set, when counting the total amount of characters ahead of time may be costly.

Note: When manipulating LOBs using LOB methods, such as the `Write` and `CopyFromFile`, the LOB object is not automatically trimmed if the length of the new data is shorter than the old data. Use the `Trim` (OraLOB) method to shrink the LOB object to the size of the new data.

Examples

Be sure that you have installed the OraLOB Schema Objects as described in "Schema Objects Used in LOB Data Type Examples" on page A-3.

Multiple-Piece Write of a LOB Example

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim PartDesc As OraClob
Dim buffer As String
Dim chunksize As Long
Dim amount_written As Long

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

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

'Create the OraDynaset Object
Set OraDynaset = OraDatabase.CreateDynaset("select * from part", 0&)
Set PartDesc = OraDynaset.Fields("part_desc").Value
chunksize = 32000

'Re adjust the buffer size
buffer = String$(chunksize, 32)
FNum = FreeFile

'Open the file.
Open "partdesc.dat" For Binary As #FNum

'set the offset and PollingAmount properties for piece wise
'Write operation
PartDesc.offset = 1
PartDesc.PollingAmount = LOF(FNum)
remainder = LOF(FNum)

'Lock the row for write operation
OraDynaset.Edit
Get #FNum, , buffer

'Do first write operation
amount_written = PartDesc.Write(buffer, chunksize, ORALOB_FIRST_PIECE)

While PartDesc.Status = ORALOB_NEED_DATA
```

```
  remainder = remainder - chunksize
  If remainder < chunksize Then
    piecetype = ORALOB_LAST_PIECE
    chunksize = remainder
   Else
    piecetype = ORALOB_NEXT_PIECE
  End If
  Get #FNum, , buffer

  amount_written = PartDesc.Write(buffer, chunksize, piecetype)
Wend

Close FNum

'call Update method to commit the transaction
OraDynaset.Update
```

Single-Piece Write of a LOB Example

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim PartImage As OraBlob
Dim buffer() As Byte

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

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

'Add PartDesc as an Output parameter and set its initial value.
OraDatabase.Parameters.Add "PartImage", Null, ORAPARM_OUTPUT
OraDatabase.Parameters("PartImage").ServerType = ORATYPE_BLOB

'Begin the transaction
OraSession.BeginTrans

'Execute the statement returning 'PartDesc'
OraDatabase.ExecuteSQL ("BEGIN select part_image into :PARTIMAGE" & _
            "from part where part_id = 1 for update NOWAIT; END;")

'Get 'PartDesc' from Parameters collection
Set PartImage = OraDatabase.Parameters("PartImage").Value

'Get a free file number
FNum = FreeFile

'Open the file.
Open "PartImage.Dat" For Binary As #FNum

'Re adjust the buffer size to hold entire file data
ReDim buffer(LOF(FNum))
Get #FNum, , buffer

'Do one write operation
amount_written = PartImage.Write(buffer)

Close FNum
MsgBox "Amount written to the LOB data is " & amount_written
```

```
'Ends the transaction
OraSession.CommitTrans
```

See Also:

- Offset (OraLOB/BFILE) Property on page 11-112
- PollingAmount Property on page 11-125
- Status (OraLOB/BFILE) Property on page 11-154
- Trim (OraLOB) Method on page 10-254
- Writing LOB Data on page 4-6

11

Server Properties

This chapter describes the Oracle Objects for OLE Server properties.

For an introduction to Server Objects, see "Oracle Objects for OLE In-Process Automation Server" on page 1-2.

This chapter contains these topics:

Server Properties: A to F

- Consumer (OraAQ) Property
- Correlate (OraAQ) Property
- Correlation (OraAQMsg) Property
- Count Property
- Count (OraMetaData) Property
- Count (OraObject/Ref) Property
- Database Property
- DatabaseName Property
- Databases Property
- Day (OraTimeStamp) Property
- Day (OraTimeStampTZ) Property
- Days Property
- DbPoolCurrentSize Property
- DbPoolInitialSize Property
- DbPoolMaxSize Property
- Delay (OraAQMsg) Property
- DequeueMode (OraAQ) Property
- DequeueMsgId (OraAQ) Property
- DirectoryName Property
- DynasetOption Property

Server Properties: E to L

- EditMode Property
- EditOption (OraRef) Property
- ElementType Property
- EOC Property
- EOF Property
- ExceptionQueue Property
- Exists Property
- Expiration (OraAQMsg) Property
- FetchLimit Property
- FetchSize Property
- FieldIndex Property
- FieldName Property
- FieldOriginalName Property
- FieldOriginalNameIndex Property
- Fields Property
- FileName Property

- Filter Property
- Format (OraNumber) Property
- Format (OraTimeStamp) Property
- Format (OraTimeStampTZ) Property
- HexValue (OraRef) Property
- Hour (OraTimeStamp) Property
- Hour (OraTimeStampTZ) Property
- Hours Property
- IsLocator (OraCollection) Property
- IsMDObject Property
- IsNull (OraCollection) Property
- IsNull (OraLOB/BFILE) Property
- IsNull (OraObject) Property
- IsOpen (OraBFILE) Property
- IsRefNull (OraRef) Property
- LastErrorText Property
- LastModified Property
- LastServerErr Property
- LastServerErrPos Property
- LastServerErrText Property

Server Properties: M to O

- MaxSize (OraCollection) Property
- MinimumSize Property
- Minute (OraTimeStamp) Property
- Minute (OraTimeStampTZ) Property
- Minutes Property
- Month (OraTimeStamp) Property
- Month (OraTimeStampTZ) Property
- Months Property
- Name Property
- Name (AQAgent) Property
- Name (OraAttribute) Property
- Name (OraMDAttribute) Property
- Nanosecond(OraTimeStamp) Property
- Nanonsecond (OraTimeStampTZ) Property
- Nanonseconds Property
- Navigation (OraAQ) Property

- NoMatch Property
- NonBlockingState Property
- Offset (OraLOB/BFILE) Property
- OIPVersionNumber Property
- Options Property
- OraIDataType Property
- OraMaxDSize Property
- OraMaxSize Property
- OraNullOK Property
- OraPrecision Property
- OraScale Property

Server Properties: P to T

- Parameters Property
- PinOption (OraRef) Property
- PollingAmount Property
- Priority (OraAQMsg) Property
- RDMSVersion Property
- RecordCount Property
- RelMsgId (OraAQ) Property
- RowPosition Property
- SafeArray (OraCollection) Property
- Second (OraTimeStamp) Property
- Second (OraTimeStampTZ) Property
- Seconds Property
- Server Property
- ServerType Property
- Session Property
- Sessions Property
- Size Property
- Size (OraCollection) Property
- Size (OraLOB and OraBFILE) Property
- SnapShot Property
- Sort Property
- SQL Property
- Status Property
- Status (OraLOB/BFILE) Property
- Subscriptions Property

- TableName (OraRef) Property
- TableSize (OraCollection) Property
- TimeZone (OraTimeStampTZ) Property
- TotalDays Property
- TotalYears Property
- Transactions Property
- Truncated Property
- Type Property
- Type (OraAttribute) Property
- Type (OraCollection) Property
- Type (OraMetaData) Property
- TypeName (OraObject and OraRef) Property

Server Properties: U to Z

- Updatable Property
- Value Property
- Value (OraAttribute) Property
- Value (OraAQMsg) Property
- Value (OraIntervalDS) Property
- Value (OraIntervalYM) Property
- Value (OraMDAttribute) Property
- Value (OraNumber) Property
- Value (OraTimeStamp) Property
- Value (OraTimeStampTZ) Property
- Version (OraObject and Ref) Property
- Visible (OraAQ) Property
- Wait (OraAQ) Property
- XMLAsAttribute Property
- XMLCollID Property
- XMLEncodingTag Property
- XMLNullIndicator Property
- XMLOmitEncodingTag Property
- XMLRowID Property
- XMLRowsetTag Property
- XMLRowTag Property
- XMLTagName Property
- XMLUpperCase Property
- Year (OraTimeStamp) Property

- Year (OraTimeStampTZ) Property
- Years Property

Address (OraAQAgent) Property

Applies To

OraAQAgent Object on page 9-5

Description

Returns a 128-byte string representing the protocol-specific address of the recipient. The format is: `[schema.]queue[@dblink]`

Usage

```
agent_address = qMsg.AQAgent.Address
```

Data Type

`String`

ArraySize Property

Applies To

OraParamArray Object on page 9-47

Description

Specifies the array size (that is, number of elements in an array) of an `OraParameter` string buffer. Not available at design time and read-only at run time.

Usage

```
OraParamArray.ArraySize
```

Data Type

```
Integer
```

Remarks

You specify the `ArraySize` during `AddTable`. See the `AddTable` method for the `OraParamArray` object.

See Also: AddTable Method on page 10-23

AutoCommit Property

Applies To

OraDatabase Object on page 9-28

Description

Returns or sets the `AutoCommit` property of the `OraDatabase` object.

Usage

```
autocommit = OraDatabase.AutoCommit
OraDatabase.AutoCommit = [ True | False
```

Data Type

Boolean

Remarks

If the `AutoCommit` property is set to `True`, all the data operations that modify data in the database are automatically committed after the statement is executed.

If the `AutoCommit` property is set to `False`, you need to use the `OraDatabase` transaction methods (`BeginTrans`, `CommitTrans`, and `Rollback`) or SQL statements to control transactions.

Examples

The following example shows how to control transactions with SQL statements after setting the `AutoCommit` property to `False`.

```
Dim session As OraSession
Dim MyDb As OraDatabase
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set MyDb = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0)
MyDb.AutoCommit = False
MyDb.ExecuteSQL ("update emp set sal = 100000" & _
                 "where ename = 'JOHN SMITH' ")
MyDb.ExecuteSQL ("commit")
```

See Also:

- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- Rollback Method on page 10-235

BOC Property

Applies To

OraCollection Object on page 9-19

Description

Indicates `True` if the collection iterator moves before the first element of a collection.

Usage

```
boc_flag = OraCollection.BOC
```

Data Type

boolean

Examples

See "Example: OraCollection Iterator" on page 10-88

BOF Property

Applies To

OraDynaset Object on page 9-30

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
```

Data Type

`Integer` (Boolean)

Remarks

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

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

Examples

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

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

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

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

```
 Do Until OraDynaset.BOF
  OraDynaset.MovePrevious
 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.MoveFirst
 'The recordset is now positioned at the first record.

End Sub
```

See Also:

- EOF Property on page 11-56
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199

Bookmark Property

Applies To

OraDynaset Object on page 9-30

Description

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

Usage

```
row_bookmark = oradynaset.Bookmark
oradynaset.Bookmark = row_bookmark
```

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.

Remarks

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

`Bookmark` objects exist only for the life of the dynaset and are specific to a particular dynaset. They cannot be shared among dynasets. However, `Bookmark` objects of a dynaset and their clones are interchangeable.

Before attempting to use `Bookmark` objects, check the `BookMarkable` property of that dynaset to see if it supports bookmarks.

Examples

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 OraClient
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset
 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.OpenDatabase("ExampleDb", "scott/tiger", 0&)
```

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

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

 'Move to the last record and display empno.
 OraDynaset.MoveLast
 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
```

See Also:

- BookMarkable Property on page 11-15
- Clone Method on page 10-52
- LastModified Property on page 11-86

BookMarkable Property

Applies To

OraDynaset Object on page 9-30

Description

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

Usage

```
if_bookmarkable = oradynaset.Bookmarkable
```

Data Type

`Integer` (Boolean)

Remarks

This property returns `True` unless the No Cache mode was set when the specified dynaset was created; otherwise, it returns `False`.

See Also:

- Bookmark Property on page 11-13
- CreateDynaset Method on page 10-85

CacheBlocks Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or set cache maximum number of blocks.

Usage

```
set blocks = oradynaset.CacheBlocks
oradynaset.CacheBlocks = blocks
```

Data Type

```
Integer
```

CacheChanged Property

Applies To

OraDynaset Object on page 9-30

Description

`True` if cache or fetch parameters have been changed.

Usage

```
set Changed = oradynaset.CacheChanged
```

Data Type

Boolean

CacheMaximumSize Property

Applies To

OraDatabase Object on page 9-28

Description

Sets the maximum size (high watermark) for the client-side object cache as a percentage of the optimal size. The default value is 10%.

Usage

```
Oradatabase.CacheMaximumSize maxsize
```

Data Type

```
Long
```

Remarks

If the memory occupied by the objects currently in the cache exceeds the high watermark (maximum object cache size), then the cache automatically begins to free unmarked objects that have a pin count of zero. The cache continues freeing those objects until memory use in the cache reaches the optimal size, or until it runs out of objects eligible for freeing.

CacheOptimalSize Property

Applies To

OraDatabase Object on page 9-28

Description

Sets the optimal size for the client-side object cache in bytes. The default value is `200` KB.

Usage

```
Oradatabase.CacheOptimalSize optimalsize
```

Data Type

`Long`

Remarks

This parameter increases the client-side object cache size. If the memory occupied by the objects currently in the cache exceeds the high watermark (maximum object cache size), then the cache automatically begins to free unmarked objects that have a pin count of zero. The cache continues freeing those objects until memory use in the cache reaches the optimal size, or until it runs out of objects eligible for freeing. This parameter should be set to an appropriate value so that object cache can accommodate all the fetched object instance from Oracle Database 10*g*. This is property is useful in performance tuning for accessing an Oracle Database 10*g* object instance.

CacheSliceSize Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets cache slice size.

Usage

```
set Slicesize = oradynaset.CacheSliceSize
oradynaset.CacheSliceSize = Slicesize
```

Data Type

`Integer`

CacheSlicesPerBlock Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets cache slices for each block.

Usage

```
set Perblock = oradynaset.CacheSlicePerBlock
oradynaset.CacheSlicePerBlock  = Perblock
```

Data Type

```
Integer
```

Client Property

Applies To

OraSession Object on page 9-58

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
```

Data Type

OLE Object (`OraClient`)

Remarks

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

See Also: OraClient Object on page 9-18

Connect Property

Applies To

OraConnection Object on page 9-27

OraDatabase Object on page 9-28

Description

Returns the user name of the connection 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
```

Data Type

`String`

Remarks

- `OraConnection.Connect`

  Returns the user name of the connection string associated with the connection.

- `OraDatabase.Connect`

  Returns the user name of the connection string associated with the specified database. It is equivalent to referencing `OraDatabase.Connection.Connect`.

The password associated with the user name is never returned.

Examples

This example demonstrates the use of the `Connect` and `DatabaseName` properties to display the user name and database name to which the user is connected. Copy and paste this code into the definition section of a form. Then, press **F5**.

```
Sub Form_Load ()

 'Declare variables
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

End Sub
```

See Also:

- OpenDatabase Method on page 10-212
- DatabaseName Property on page 11-37

Connection Property

Applies To

OraDatabase Object on page 9-28

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

Description

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

Usage

```
Set oraconnection = oradatabase.Connection
Set oraconnection = oradynaset.Connection
Set oraconnection = orasqlstmt.Connection
```

Data Type

OLE Object (`OraConnection`)

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. This is equivalent to referencing `oradynaset.Database.Connection`.

- `OraSQLStmt`.Connection

  Returns the connection object associated with this `OraSQLStmt` object. This is equivalent to referencing `orasqlstmt.Database.Connection`.

See Also: OraConnection Object on page 9-27

ConnectionOK Property

Applies To

OraDatabase Object on page 9-28

OraConnection Object on page 9-27

Description

Returns a Boolean value indicating the status of the database connection associated with the `OraConnection` object. A return value of `True` implies that the connection is alive in the connection object associated with the specified database. If the connection has been dropped, this property returns `False`.

Not available at design time and read-only at run time.

Usage

```
ConnectionStat = OraDatabase.ConnectionOK
ConnectionStat = OraDatabase.Connection.ConnectionOK
```

Data Type

Boolean

Remarks

- `OraDatabase.ConnectionOK`

  Returns the connection status of 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.

- `OraConnection.ConnectionOK`

  Returns the status of the underlying connection to the database. This is equivalent to `OraDatabase.OraConnection.ConnectionOK`.

  See Also:

  - Connection Property on page 11-25
  - "Detection of Lost Connections" on page 3-9

Connections Property

Applies To

OraSession Object on page 9-58

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
```

Data Type

OLE Object (`OraParameters`)

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 indexes and invalid names return a `Null OraConnection` object.

See Also:

- Count Property on page 11-31
- OraConnection Object on page 9-27
- OraConnections Collection on page 9-66

Consumer (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Applicable only for a dequeue operation.

Usage

```
Q.Consumer = consumer_name
```

Data Type

```
String
```

Remarks

The value is a string representing the name of the consumer. Only those messages matching the consumer name are accessed.

Examples

```
Dim DB As OraDatabase
    Dim Q as OraAQ
      set Q = DB.CreateAQ("Q_MSG_MULTIPLE")
      'Dequeue only message meant for ANDY
       Q.consumer = "ANDY"
      'other processing...
       Q.Dequeue
```

Correlate (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies the identification to look for while dequeuing messages.

Usage

```
Q.Correlate = "RELATIVE_MESSAGE_ID"
```

Data Type

`String`

Remarks

Applicable only for a dequeue operation.

Correlation (OraAQMsg) Property

Applies To

OraAQMsg Object on page 9-6

Description

Specifies the identification for the message. This can then be used as a means of dequeuing specific messages.

Usage

```
Msg.Correlation = my_message
```

Data Type

```
String
```

Remarks

Applicable only for a message that is being enqueued. Returns any string up to 128 bytes.

See Correlate for dequeuing using this identifier.

See Also: Correlate (OraAQ) Property on page 11-29

Count Property

Applies To

OraConnections Collection on page 9-66

OraFields Collection on page 9-67

OraParameters Collection on page 9-68

OraSessions Collection on page 9-69

OraSubscriptions Collection on page 9-70

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
subscriptions_count = OraSubscriptions.Count
```

Data Type

`Integer`

Remarks

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

Examples

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 OraClient
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Create the OraDynaset Object.
```

```
 Set OraDynaset = OraDatabase.CreateDynaset("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
```

See Also:

- OraConnection Object on page 9-27
- OraField Object on page 9-33

Count (OraMetaData) Property

Applies To

OraMetaData Object on page 9-39

Description

An integer representing the number of `OraMDAttribute` objects contained in this collection.

Usage

```
count = OraMetaData.Count
```

Data Type

`Integer`

See Also: OraMDAttribute Object on page 9-38

Count (OraObject/Ref) Property

Applies To

OraObject Object on page 9-43

OraRef Object on page 9-52

Description

Returns the number of `OraAttribute` objects in the collection. This is same as the total number of attributes of the underlying referenceable object of `OraRef` or underlying value instance of `OraObject`. Read-only at run time.

Usage

```
attrcount = OraRef.Count
attrcount = OraObject.Count
```

Data Type

`Integer`

Remarks

Individual attributes can be accessed by using a subscript or the name of the attribute. The `OraObject` or `OraRef` attribute index starts at `1`.

Examples

The following example shows the use of the `Count` property. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Address as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from  person_tab",0&)

'retrieve a address column from person_tab.
'Here Value property of OraField object returns Address OraObject
set Address = OraDynaset.Fields("Addr").Value

'access the attribute by dot notation
msgbox Address.Street

'access the attribute using '!' notation ( early binding application)
```

```
msgbox Address!Street

'access the attribute by index
msgbox Address(1)

'access the attribute by name
msgbox Address("Street")

'access all the attributes of Address OraObject in the dynaset
Do Until OraDynaset.EOF
    For index = 1 To Address.Count
        msgbox Address(index)
    Next Index
  OraDynaset.MoveNext

Loop
```

See Also: OraAttribute Object on page 9-7

Database Property

Applies To

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

Description

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

Usage

```
Set oradatabase = oradynaset.Database
Set oradatabase = orasqlstmt.Database
```

Data Type

OLE Object (`OraDatabase`)

Remarks

The `OraDynaset.Database` property returns the `OraDatabase` object from which the specified dynaset was created.

The `OraSQLStmt.Database` property returns the `OraDatabase` object from which the specified `SQLStmt` object was created.

See Also:

- CreateDynaset Method on page 10-85
- OraDatabase Object on page 9-28

DatabaseName Property

Applies To

OraConnection Object on page 9-27

OraDatabase Object on page 9-28

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
```

Data Type

`String`

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`.

Examples

This example demonstrates the use of the `Connect` and `DatabaseName` properties to display the user name 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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

End Sub
```

See Also:

- Connect Property on page 11-23
- Connection Property on page 11-25
- OpenDatabase Method on page 10-212

Databases Property

Applies To

OraServer Object on page 9-56

Description

Returns a collection interface containing all user sessions that have been established using this object.

Usage

```
Set myCollection = oraserver.Databases
```

Data Type

OLE Object (`OraCollection`)

See Also: OraCollection Object on page 9-19

Day (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Gets and sets the `Day` attribute of an `OraTimeStamp` object.

Usage

```
day= OraTimeStampObj.Day
OraTimeStampObj.Day= day
```

Arguments

Arguments	Description
[in] *day*	The `Day` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Day (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Day` attribute of an `OraTimeStampTZ` object.

Usage

```
day= OraTimeStampTZObj.Day
OraTimeStampTZObj.Day= day
```

Arguments

Arguments	Description
[in] *day*	The `Day` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Days Property

Applies To

OraIntervalDS Object on page 9-35

Description

Gets and sets the `Days` attribute of an `OraIntervalDS` object

Usage

```
days = OraIntervalDSObj.Days
OraIntervalDSObj.Days = days
```

Arguments

Arguments	Description
[in] *days*	An `Integer` specifying the value of the `Days` attribute of the `OraIntervalDS` object.

Data Type

`Integer`

DbPoolCurrentSize Property

Applies To

OraSession Object on page 9-58

Description

Contains the number of currently active database objects in the pool. It is a read-only property.

Usage

```
curr_size = OraSession.DbPoolCurrentSize
```

Data Type

```
Integer
```

Remarks

An active database object in the pool that contains a live connection to the database.

DbPoolInitialSize Property

Applies To

OraSession Object on page 9-58

Description

Contains the initial size of the pool. It is a read-only property.

Usage

```
init_size = OraSession.DbPoolInitialSize
```

Data Type

```
Integer
```

DbPoolMaxSize Property

Applies To

OraSession Object on page 9-58

Description

Contains the maximum pool size. It is a read-only property.

Usage

```
max_size = OraSession.DbPoolMaxSize
```

Data Type

```
Integer
```

Delay (OraAQMsg) Property

Applies To

OraAQMsg Object on page 9-6

Description

Specifies the number of seconds to delay this enqueued message. Set this property to delay the immediate consumption of the message.

Usage

```
Msg.Delay = seconds
```

Data Type

```
Integer
```

Remarks

Applicable only for a message that is enqueued.

This delay represents the number of seconds after which the message is available for dequeuing.

Possible values are:

- Any valid positive integer.
- `ORAAQ_MSG_NO_DELAY`

  Default is 0 seconds. The message is available immediately.

DequeueMode (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies the locking behavior associated with the dequeue operation.

Usage

```
Q.DequeueMode = locking_mode
```

Data Type

```
Integer
```

Remarks

Possible values are:

- `ORAAQ_DQ_BROWSE` (`1`)

  Does not lock the message when dequeuing.

- `ORAAQ_DQ_LOCKED` (`2`)

  Reads and obtains a write lock on the message.

- `ORAAQ_DQ_REMOVE` (`3`)(Default)

  Reads the message, and updates or deletes it.

DequeueMsgId (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Returns an array of raw bytes, specifying the message identifier of the message to be dequeued.

Usage

```
Q.DequeueMsgid = msg_id
```

Data Type

```
String
```

Remarks

Applicable only for a dequeue operation.

DirectoryName Property

Applies To

OraBFILE Object on page 9-9

Description

Gets or sets the directory alias name.

Usage

```
diralias = OraBFile.DirectoryName
OraBFile.DirectoryName = diralias
```

Arguments

Arguments	Description
[in] [out] *diralias*	A `String` specifying the directory name to be retrieved or set.

Data Type

`String`

Remarks

This `String` is case-sensitive.

DynasetOption Property

Applies To

OraParameter Object on page 9-50

Description

Specifies the dynaset option for a dynaset created from the PL/SQL cursor.

Usage

```
oraparameter.DynasetOption = dyn_opts
```

Remarks

This property should be called before executing a PL/SQL procedure containing a cursor variable. By default, the dynaset is created with the `ORADYN_READONLY` option.

The possible values are:

Possible Values	Value	Description
`ORADYN_DEFAULT`	`&H0&`	Accepts the default behavior.
`ORADYN_NO_BLANKSTRIP`	`&H2&`	Does not remove trailing blanks from character string data retrieved from the database.
`ORADYN_NOCACHE`	`&H8&`	Does 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.
`ORADYN_NO_MOVEFIRST`	`&H40&`	Does not force a `MoveFirst` operation when a dynaset is created. `BOF` and `EOF` are both `True`.

See Also:

- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- BOF Property on page 11-11
- EOF Property on page 11-56

EditMode Property

Applies To

OraDynaset Object on page 9-30

Description

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

Usage

```
edit_mode = oradynaset.EditMode
```

Data Type

```
Integer
```

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 *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\oo4o\oraconst.txt` file and are intended to match similar constants in the Visual Basic `constant.txt` file.

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

See Also:

- AddNew Method on page 10-21
- Edit Method on page 10-134
- Update Method on page 10-257

EditOption (OraRef) Property

Applies To

OraRef Object on page 9-52

Description

Specifies whether the object is to be locked during the pin operation.

Usage

```
edit_option = OraRef.EditOption
OraRef.EditOption = edit_option
```

Arguments

Arguments	Description
[in] [out] *edit_option*	An `Integer` representing the edit option.

Data Type

Integer

Remarks

This property should be called before a pin operation on a `Ref` value, before accessing an attribute for the first time on the `OraRef` object. This option is useful if the object attributes are modified immediately after the pin operation. Locking the object instance during the pin operation saves the round-trip to the database during the `Edit` (`OraRef`) operation.

Possible values of `edit_option` are:

Constant	Value	Description
`ORAREF_NO_LOCK`	1	Does not lock the object in the database (default).
`ORAREF_EXCLUSIVE_LOCK`	2	Maintains an exclusive lock on the object in the database.
`ORAREF_NOWAIT_LOCK`	3	Maintains an exclusive lock on the object in the database with the nowait option.

Examples

The following example shows the usage of the `EditOption` property. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
```

```
Dim OraDynaset as OraDynaset
Dim Person as OraRef

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

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

'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers", 0&)

'retrieve a aperson column from customers. Here Value property of OraField object

'returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'set the ORAREF_EXCLUSIVE_LOCK EditOption on the Person object.
Person.EditOption = ORAREF_EXCLUSIVE_LOCK

'pin the Person Ref. This operation also locks the underlying
'referenceable 'object in the server
MsgBox Person.Name

'call Edit method on Person OraRef.
'This method does not make any network round-trip

Person.Edit
Person.Name = "Eric"
Person.Age = 35
Person.Update
```

See Also: Edit (OraRef) Method on page 10-136

ElementType Property

Applies To

OraCollection Object on page 9-19

Description

An integer code representing the server type of an element. This property is read-only at run time.

Usage

```
elem_type = OraCollection.ElementType
```

Data Type

```
Integer
```

Remarks

The codes correspond to the Oracle external data types. The following Oracle element 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_VARCHAR	9	VARCHAR
ORATYPE_DATE	12	DATE
ORATYPE_UINT	68	UNSIGNED INTEGER
ORATYPE_CHAR	96	CHAR
ORATYPE_CHARZ	97	Null Terminated CHAR
ORATYPE_BFLOAT	100	BINARY_FLOAT
ORATYPE_BDOUBLE	101	BINARY_DOUBLE
ORATYPE_OBJECT	108	Object
ORATYPE_REF	110	Ref

EOC Property

Applies To

OraCollection Object on page 9-19

Description

Returns `True` if the collection iterator moves past the last element of a collection.

Usage

```
eoc_flag = OraCollection.EOC
```

Data Type

Boolean

Examples

See "Example: OraCollection Iterator" on page 10-88.

EOF Property

Applies To

OraDynaset Object on page 9-30

Description

Indicates 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
```

Data Type

`Integer` (Boolean)

Remarks

Returns `True` if an attempt has been made to move after the last record in the dynaset using the `MoveNext` method. Otherwise, returns `False`.

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

Examples

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
 Dim OraSession As OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

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

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

```
 Do Until OraDynaset.BOF
  OraDynaset.MovePrevious
 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.MoveFirst

 'The recordset is now positioned at the first record.

End Sub
```

See Also:

- BOF Property on page 11-11
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199

ExceptionQueue Property

Applies To

OraAQMsg Object on page 9-6

Description

Specifies the name of the queue to which message should be moved if it cannot be processed successfully.

Usage

```
Msg.ExceptionQueue queue_name
```

Data Type

```
String
```

Remarks

Applicable only for a message that is being enqueued.

Possible values are:

- A `String` containing a valid queue name
- `Null` (Default)

A message is moved to the exception queue if the number of dequeue attempts has expired or has exceeded *max_retries* specified in the `DBMS_AQADM.CREATE_QUEUE` command.

Exists Property

Applies To

OraBFILE Object on page 9-9

Description

Returns `True` if the `OraBFILE` points to a `BFILE` that exists on the database.

Usage

```
exists = OraBFile.Exists
```

Data Type

Boolean

Remarks

Read privileges on the directory where the `BFILE` is located are required to use this property. The operating system-specific permissions must have been set for the directory to make sure that the user can read the directory.

Appropriate privileges must be set up in the database previously. For example, to ensure that a user (`scott`) can read a directory (`BfileDirectory`) through the `Exists` property, the following SQL statement must be executed:

```
GRANT READ ON DIRECTORY BfileDirectory TO scott;
```

Expiration (OraAQMsg) Property

Applies To

OraAQMsg Object on page 9-6

Description

Specifies, in seconds, the time for which the message is available for dequeuing.

Usage

```
Msg.Expiration = seconds
```

Data Type

```
Integer
```

Remarks

This property is an offset from the delay. It is applicable only for a message that is being enqueued.

Possible Values are:

- Any integer.
- `ORAAQ_MSG_NO_XPIRE` (0)

  Default `0` - The message will never expire.

FetchLimit Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets the array size of the fetch.

Usage

```
set Limit = oradynaset.FetchLimit
oradynaset.FetchLimit = Limit
```

Data Type

`Integer`

FetchSize Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets the array buffer size of the fetch.

Usage

```
set Size = oradynaset.FetchSize
oradynaset.FetchSize = Size
```

Data Type

```
Integer
```

FieldIndex Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the index of the field indicated by the *field_name* argument.

Usage

```
set index = oradynaset.FieldIndex(field_name)
```

Arguments

Arguments	Description
[in] *field_name*	The name of the field as it appears in the SQL statement that the dynaset used most recently.

Data Type

Integer

Remarks

Accessing fields of a dynaset using an index is more efficient than accessing them by name. If you need to access a particular field many times, use this method to translate its name into its index.

See Also:

- FieldName Property on page 11-64
- FieldOriginalName Property on page 11-65
- FieldOriginalNameIndex Property on page 11-66

FieldName Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the field name in the `SELECT` statement in the dynaset.

Usage

```
set field_name = oradynaset.FieldName(index)
```

Arguments

Arguments	Description
[in] *index*	Index of the name of the field as it appears in the SQL statement.

Data Type

`String`

See Also:

- FieldIndex Property on page 11-63
- FieldOriginalName Property on page 11-65
- FieldOriginalNameIndex Property on page 11-66

FieldOriginalName Property

Applies To

OraDynaset Object on page 9-30

Description

Gets the original field name used in the `SELECT` statement in the dynaset.

Usage

```
set field_name = oradynaset.FieldOriginalName(index)
```

Arguments

Arguments	Description
[in] *index*	An `Integer` specifying the field index of the original field name as it appears in the SQL statement.

Data Type

`String`

Remarks

The `FieldOriginalName` property returns a string containing the original column name specified in the SQL statement during dynaset creation. This property is useful when a SQL statement contains `SCHEMA.TABLE.COL` as the name of the field. This enables duplicate column names to be referenced. Another way to avoid duplicate columns is to specify an alias in the SQL statement.

See Also:

- FieldIndex Property on page 11-63
- FieldName Property on page 11-65
- FieldOriginalNameIndex Property on page 11-66

FieldOriginalNameIndex Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the index of the field indicated by the original field name used in the SQL `SELECT` statement.

Usage

```
set index = oradynaset.FieldOriginalNameIndex(name)
```

Arguments

Arguments	Description
[in] *name*	The original name of the field as it appears in the SQL statement.

Data Type

`Integer`

Remarks

Accessing fields of a dynaset by index is more efficient than accessing them by name. If you need to access a particular field many times, use this method to translate its original name into its index.

See Also:

- FieldIndex Property on page 11-63
- FieldName Property on page 11-65
- FieldOriginalName Property on page 11-65

Fields Property

Applies To

OraDynaset Object on page 9-30

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
```

Data Type

OLE Object (`OraFields`)

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 is not within the collection (`0` to `Count - 1`) results in the return of a `Null OraField` object.

See Also:

- Count Property on page 11-31
- OraFields Collection on page 9-67

FileName Property

Applies To

OraBFILE Object on page 9-9

Description

Gets or sets a filename. Read and write at run time.

Usage

```
filename  = OraBFile.FileName
OraBFile.FileName = filename
```

Arguments

Arguments	Description
[in] [out] *filename*	A `String` specifying the directory name to be retrieved or set.

Data Type

`String`

Remarks

This string can be case-sensitive depending on the database operating system.

See Also: OraField Object on page 9-33

Filter Property

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 on page 10-85
- OraDynaset Object on page 9-30
- OraFields Collection on page 9-67
- OraParameter Object on page 9-50

Format (OraNumber) Property

Applies To

OraNumber Object on page 9-41

Description

The format string used in `OraNumber` operations. For details about format strings, see *Oracle Database SQL Quick Reference*. Read and write at run time.

Usage

```
OraNumber.Format = formatstring
formatstring = OraNumber.Format
```

Arguments

Arguments	Description
[in] *formatstring*	A format string used in `OraNumber` operations.

Data Type

`String`

Remarks

An error is returned if the format string is set to an invalid value. To reset the format to the default, set it to an empty string.

See Also: *Oracle Database SQL Quick Reference*

Format (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the TIMESTAMP format used to display the OraTimeStamp object as a string.

Usage

```
format = OraTimeStampObj.Format
OraTimeStampObj.Format = format
```

Arguments

Arguments	Description
[in] *format*	The format used to display an OraTimeStamp object as a string.

Data Type

String

Remarks

If Format is Null, the session TIMESTAMP format is used to display the OraTimeStamp object as a string.

Format (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the TIMESTAMP WITH TIME ZONE format used to display the OraTimeStampTZ object as a string.

Usage

```
format = OraTimeStampTZObj.Format
OraTimeStampTZObj.Format = format
```

Arguments

Arguments	Description
[in] *format*	The format used to display an OraTimeStampTZ object as a string.

Data Type

String

Remarks

If Format is Null, the session TIMESTAMP WITH TIME ZONE format is used to display the OraTimeStampTZ object as a string.

HexValue (OraRef) Property

Applies To

OraRef Object on page 9-52

Description

Returns the hexidecimal value of the `REF`.

Usage

```
hexstring = OraRef.HexValue
```

Remarks

The hexidecimal value of the `REF` can be used by the `OraDatabase.FetchOraRef` method.

See Also: FetchOraRef Method on page 10-149

Hour (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Hour` attribute of an `OraTimeStamp` object.

Usage

```
hour = OraTimeStampObj.Hour
OraTimeStampObj.Hour = hour
```

Arguments

Arguments	Description
[in] *hour*	The `Hour` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Hour (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Hour` attribute of an `OraTimeStampTZ` object.

Usage

```
hour = OraTimeStampTZObj.Hour
OraTimeStampTZObj.Hour = hour
```

Arguments

Arguments	Description
[in] *hour*	The `Hour` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Hours Property

Applies To

OraIntervalDS Object on page 9-35

Description

Gets and sets the `Hours` attribute of an `OraIntervalDS` object.

Usage

```
hours = OraIntervalDSObj.Hours
OraIntervalDSObj.Hours = hours
```

Arguments

Arguments	Description
[in] *hours*	An `Integer` specifying the value of the `Hours` attribute of the `OraIntervalDS` object.

Data Type

`Integer`

IsLocator (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns `True` if the collection instance of the `OraCollection` object is locator-based; otherwise, returns `False`.

Usage

```
islocator = OraCollection.IsLocator
```

Data Type

`Integer` (Boolean)

IsMDObject Property

Applies To

OraMDAttribute Object

Description

Returns `True` if the `Value` property is another `OraMetaData` object; otherwise, the property is `False`.

Usage

```
isobject = OraMDAttribute.IsMDObject
```

Data Type

Boolean

IsNull (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns `True` if the collection value of the `OraCollection` object is `Null`.

Usage

```
isnull = OraObject.IsNull
```

Data Type

`Integer` (Boolean)

Remarks

Accessing elements of a `Null` collection results in an error. The `IsNull` property should be checked before accessing elements of an underlying collection.

IsNull (OraLOB/BFILE) Property

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Returns `True` if the LOB or `BFILE` refers to a `Null` value in the database; otherwise, returns `False`. This property is read-only.

Usage

```
IsNull = OraBfile.IsNull
IsNull = OraBlob.IsNull
IsNull = OraClob.IsNull
```

Data Type

Boolean

Remarks

Some LOB or `BFILE` properties and methods are not valid when a LOB or `BFILE` is `Null`.

This property makes it possible to check for `Null` values and avoid these errors.

IsNull (OraObject) Property

Applies To

OraObject Object on page 9-43

Description

Returns `True` if underlying value instance of the `OraObject` object is `Null`. Read-only at run time.

Usage

```
isnull = OraObject.IsNull
```

Data Type

`Integer` (Boolean)

Remarks

Accessing attributes of a `Null` value instance results in an error. The `IsNull` property can be checked before accessing attributes of an underlying value instance.

Examples

The following example shows the use of the `IsNull` property. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Address as OraObject
Dim AddressClone as OraObject

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

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

'create a dynaset object from person_tab
set OraDynaset = OraDatabase.CreateDynaset("select * from person_tab", 0&)

' insert a Null Address value instance in the table
OraDynaset.AddNew
OraDynaset.Fields("Name").value = "Eric"
OraDynaset.Fields("Addr").Value  = Null
OraDynaset.update

'move to the newly added value instance
OraDynaset.MoveLast

'retrieve a address column from person_tab. This Address object points to Null
' value instance
```

```
set Address = OraDynaset.Fields("Addr").Value

'try to access attributes of Address. the following line will result an error
msgbox Address.Street    '---------ERROR------------'

'use the IsNull property to check the nullstatus
If Address.IsNull = False Then
    MsgBox Address!Street
End if
```

IsOpen (OraBFILE) Property

Applies To

OraBFILE Object on page 9-9

Description

Returns `True` if the `OraBFILE` object is open.

Usage

```
IsOpen = OraBFile.IsOpen
```

Data Type

Boolean

Remarks

The *openness* of an object `OraBFILE` is local to this `OraBFILE` object. If two `OraBFILE` objects point to the same `BFILE` in the database, and one `OraBFILE` object calls the `Open` method and the other does not, one `OraBFILE` object will return `True` for the `IsOpen` property. The other will return `False`.

See Also: Open (OraBFILE) Method on page 10-211

IsRefNull (OraRef) Property

Applies To

OraRef Object on page 9-52

Description

Returns `True` if the underlying `Ref` value of the `OraRef` object is `Null`.

Usage

```
isnull = OraRef.IsRefNull
```

Data Type

`Integer` (Boolean)

Remarks

Accessing the attributes of a `Null Ref` value results in an error. The `IsRefNull` property should be checked before accessing attributes of an underlying referenceable object. This property is read-only at run time.

LastErrorText Property

Applies To

OraParamArray Object on page 9-47

Description

Gets the last error message. Not available at design time and read-only at run time.

Usage

```
OraParamArray.LastErrorText
```

Data Type

```
String
```

LastModified Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the `Bookmark` object 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
```

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.

Remarks

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

See Also:

- AddNew Method on page 10-21
- Bookmark Property on page 11-13
- Edit Method on page 10-134

LastServerErr Property

Applies To

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

Returns the last nonzero 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
```

Data Type

Long Integer

Remarks

This property represents the last nonzero return value from an Oracle Call Interface (OCI) database function, or zero if no error has occurred since the last `LastServerErrReset` request. For efficiency, only nonzero return values are returned; therefore, a nonzero 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 the `LastServerErr` method).

- `Orasession.LastServerErr`

  Returns all errors related to connections, such as errors on `OpenDatabase`, `BeginTrans`, `CommitTrans`, `Rollback`, and `ResetTrans` method.

- `Oradatabase.LastServerErr`

  Returns all errors related to an Oracle cursor, such as errors on dynasets and from `ExecuteSQL` method.

Examples

This example demonstrates the use of the `CreateDynaset` method and the `LastServerErr` and `LastServerErrText` properties to determine whether an Oracle error has occurred, 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Attempt to Create the OraDynaset Object.
 'Notice that the FROM keyword is missing from the SQL statement.
 Set OraDynaset = OraDatabase.CreateDynaset("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
```

See Also:

- ExecuteSQL Method on page 10-144
- LastServerErrReset Method on page 10-189
- LastServerErrText Property on page 11-90
- OpenDatabase Method on page 10-212
- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- Rollback Method on page 10-235
- ResetTrans Method on page 10-233

LastServerErrPos Property

Applies To

OraDatabase Object on page 9-28

Description

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

Usage

```
error_pos = oradatabase.LastServerErrPos
```

Data Type

```
Integer
```

Remarks

The `LastServerErrPos` property 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 the `CreateDynaset` or `ExecuteSQL` method).

See Also:

- CreateDynaset Method on page 10-85
- ExecuteSQL Method on page 10-144
- LastServerErr Property on page 11-87
- LastServerErrText Property on page 11-90

LastServerErrText Property

Applies To

OraDatabase Object on page 9-28

OraSession Object on page 9-58

Description

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

Usage

```
error_text = orasession.LastServerErrText
error_text = oradatabase.LastServerErrText
```

Data Type

`String`

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` property.
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 no additional information was available.

Examples

This example demonstrates the use of the `CreateDynaset` method and the `LastServerErr` and `LastServerErrText` properties to determine whether an Oracle error has occurred and to display the error message. 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.OpenDatabase("ExampleDb", "scott/tiger", 0&)

 'Attempt to Create the OraDynaset Object.
 'Notice that the FROM keyword is missing from the SQL statement.
```

```
 Set OraDynaset = OraDatabase.CreateDynaset("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
```

See Also:

- LastServerErr Property on page 11-87
- LastServerErrReset Method on page 10-189
- CreateDynaset Method on page 10-85

MaxSize (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns the maximum size of the collection.

Usage

```
max_size  = OraCollection.MaxSize
```

Data Type

`Integer`

Remarks

For an `OraCollection` object of type `ORATYPE_TABLE`, this property returns the current size of the collection including deleted elements. For an `OraCollection` object of type `ORATYPE_VARRAY`, the property returns the maximum size of the collection.

MinimumSize Property

Applies To

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

Description

Returns the minimum size of an `OraParameter` or `OraParamArray` string buffer or `ByteArray` (for `ORATYPE_RAW_BIN`). For `OraParamArray` objects, the minimum size property is read-only at run time. For `OraParameter` objects, the minimum size is read/write at run time.

Usage

```
oraparameter.MinimumSize
oraparamarray.MinimumSize
```

Data Type

`Integer`

Remarks

This property gets the minimum number of characters or bytes to be allocated for each element of the array. For `OraParamArray` objects, the size is specified using the `AddTable` method.

Examples

Note: This example needs the following to be run: a PL/SQL procedure called `EmployeeLong` with a `GetEmpName` procedure that uses a table with the column name `ENAME_LONG` that returns a long `ename` of approximately 200 characters.

```
Sub Form_Load ()

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

' 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", 9999, ORAPARM_INPUT

' Add ENAME as an Output parameter and set its initial value.
OraDatabase.Parameters.Add "ENAME_LONG", "foo", ORAPARM_OUTPUT
OraDatabase.Parameters("ENAME_LONG").MinimumSize = 201
 'Since we require to hold a value of more than 128 bytes

' Execute the Stored Procedure Employee.GetEmpName to retrieve ENAME_LONG.
```

```
OraDatabase.ExecuteSQL ("Begin EmployeeLong.GetEmpName (:EMPNO," & _
            "NAME_LONG); end;")
```

See Also:

- Add Method on page 10-8
- AddTable Method on page 10-23
- ExecuteSQL Method on page 10-144

Minute (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Minute` attribute of an `OraTimeStamp` object.

Usage

```
minute = OraTimeStampObj.Minute
OraTimeStampObj.Minute = minute
```

Arguments

Arguments	Description
[in] *minute*	The `Minute` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Minute (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Minute` attribute of an `OraTimeStampTZ` object.

Usage

```
minute = OraTimeStampTZObj.Minute
OraTimeStampTZObj.Minute = minute
```

Arguments

Arguments	Description
[in] *minute*	The `Minute` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Minutes Property

Applies To

OraIntervalDS Object on page 9-35

Description

Gets and sets the `Minutes` attribute of an `OraIntervalDS` object.

Usage

```
minutes = OraIntervalDSObj.Minutes
OraIntervalDSObj.Minutes = minutes
```

Arguments

Arguments	Description
[in] *minutes*	An `Integer` specifying the value of the `Minutes` attribute of the `OraIntervalDS` object.

Data Type

`Integer`

Month (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Month` attribute of an `OraTimeStamp` object.

Usage

```
month = OraTimeStampObj.Month
OraTimeStampObj.Month = month
```

Arguments

Arguments	Description
[in] *month*	The `Month` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Month (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Month` attribute of an `OraTimeStampTZ` object.

Usage

```
month = OraTimeStampTZObj.Month
OraTimeStampTZObj.Month = month
```

Arguments

Arguments	Description
[in] *month*	The `Month` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Months Property

Applies To

OraIntervalYM Object on page 9-37

Description

Gets and sets the `Months` attribute of an `OraIntervalYM` object.

Usage

```
months = OraIntervalYMObj.Months
OraIntervalYMObj.Months = months
```

Arguments

Arguments	Description
[in] *month*	An `Integer` specifying the value of the `Months` attribute of the `OraIntervalYM` object.

Data Type

`Integer`

Name Property

Applies To

OraClient Object on page 9-18

OraField Object on page 9-33

OraParameter Object on page 9-50

OraSession Object on page 9-58

OraParamArray Object on page 9-47

OraServer Object on page 9-56

OraSubscription Object on page 9-61

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
paramarray_name = oraparamarray.Name
session_name = orasession.Name
server_name = oraserver.Name
subscription_name = orasubscription.Name
```

Data Type

`String`

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 use returns the name of the field as it appears in the database. 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 SQL `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.

- `oraparamarray.Name`

  Returns the name of the specified `OraParamArray` 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.

- `oraserver.Name`

  Returns the name of the physical connection of the specified `OraServer` object.

- `orasubscription.Name`

  Returns the name used to represent the subscription. `Name` here refers to the subscription name in the form of the string `'SCHEMA.QUEUE'` if the registration is for a single consumer queue and `'SCHEMA.QUEUE:CONSUMER_NAME'` if the registration is for a multiple consumer queue.

See Also:

- CreateSession Method on page 10-109
- *Oracle Database Concepts* for more information about Oracle Database events

Name (AQAgent) Property

Applies To

OraAQAgent Object on page 9-5

Description

Returns a 30-byte string representing the name of agent.

Usage

```
agent_name = qMsg.AQAgent.Name
```

Data Type

```
String
```

Name (OraAttribute) Property

Applies To

OraAttribute Object on page 9-7

Description

A `String` containing the name of the attribute.

Usage

```
name  = OraAttribute.Name
```

Data Type

`String`

Remarks

Read-only at run time.

See Also: OraAttribute Object on page 9-7

Name (OraMDAttribute) Property

Applies To

OraMDAttribute Object on page 9-38

Description

A `String` containing the name of the attribute.

Usage

```
name = OraMDAttribute.Name
```

Data Type

`String`

Nanosecond(OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Nanosecond` attribute of an `OraTimeStamp` object.

Usage

```
nanosecond = OraTimeStampObj.Nanosecond
OraTimeStampObj.Nanosecond= nanosecond
```

Arguments

Arguments	Description
[in] *nanosecond*	The `Nanosecond` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Nanonsecond (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Nanosecond` attribute of an `OraTimeStampTZ` object.

Usage

```
nanosecond = OraTimeStampTZObj.Nanosecond
OraTimeStampTZObj.Nanosecond= nanosecond
```

Arguments

Arguments	Description
[in] *nanosecond*	The `Nanosecond` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Nanonseconds Property

Applies To

OraIntervalYM Object on page 9-37

Description

Gets and sets the `Nanoseconds` attribute of an `OraIntervalDS` object.

Usage

```
nanoseconds = OraIntervalDSObj.Nanoseconds
OraIntervalDSObj.Nanoseconds = nanoseconds
```

Arguments

Arguments	Description
[in] *nanoseconds*	An `Integer` specifying the value of the `Nanoseconds` attribute of the `OraIntervalDS` object.

Data Type

`Integer`

Navigation (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies the position of the message that will be retrieved.

Usage

```
Q.Navigation = position
```

Data Type

```
Integer
```

Remarks

Possible values are:

- `ORAAQ_DQ_FIRST_MSG` (`1`)

  Retrieves the first message that is available and matches the search criteria.

- `ORAAQ_DQ_NEXT_TRANS` (`2`)

  Skips the remainder of the current transaction group, if any, and retrieves the first message of the next transaction group. Used only if message grouping is enabled for the queue.

- `ORAAQ_DQ_NEXT_MSG` (`3`) (Default)

  Retrieves the next message that is available and matches the search criteria.

NoMatch Property

Applies To

OraDynaset Object on page 9-30 using the Address (OraAQAgent) Property Property

Description

Returns `True` if the last call to the `FindFirst`, `FindLast`, `FindNext`, or `FindPrevious` method failed.

Usage

```
Set nomatch_status = oradynaset.NoMatch
```

Data Type

Boolean

See Also: FindFirst, FindLast, FindNext, and FindPrevious Methods on page 10-151

NonBlockingState Property

Applies To

"OraSQLStmt Object" on page 9-60 created with ORASQL_NONBLK option.

Description

Returns the status of the currently executing SQL as follows:

- ORASQL_STILL_EXECUTING

  If operation is still underway.

- ORASQL_SUCCESS

  If operation has completed successfully.

Any failures are thrown as exceptions.

The application can access the output parameters, if any, as in the blocking case, after successful execution of the SQL statement.

Usage

```
status = OraSQL.NonBlockingState
if status = ORASQL_STILL_EXECUTING
    MsgBox "Still in execution"
else
    MsgBox "Execution completed successfully"
```

Return Values

ORASQL_STILL_EXECUTING(-3123) or ORASQL_SUCCESS(0)

Errors are thrown as exceptions.

See Also:

- CreateSQL Method on page 10-111
- "Asynchronous Processing" on page 3-16

Offset (OraLOB/BFILE) Property

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Gets or sets the 1-based offset into the LOB or `BFILE` for the next `Read` or `Write` operation. This property is read/write at run time.

Usage

```
offsetbytes = OraBFile.Offset
OraBFile.Offset = offsetbytes

offsetbytes = OraBlob.Offset
OraBlob.Offset = offsetbytes

offsetchars = OraClob.Offset
OraClob.Offset = offsetchars
```

Data Type

`Integer`

Remarks

This value is expressed in bytes for `OraBLOB` and `OraBFILE` or characters for the `OraCLOB` object. The default value is `1`. Setting this value to `0` raises an error. When the `PollingAmount` property is not `0` (polling is enabled), the `Offset` property can only be set before the first `Read` or `Write` operation, or after the current polling operation has completed.

See Also: PollingAmount Property on page 11-125

OIPVersionNumber Property

Applies To

OraSession Object on page 9-58

Description

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

Usage

```
version_number = orasession.OIPVersionNumber
```

Data Type

`String`

Remarks

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

Options Property

Applies To

OraDatabase Object on page 9-28

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

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
options = orasqlstmt.Options
```

Data Type

`Long Integer`

Remarks

See the `OpenDatabase` method for a description of the possible values of `oradatabase.Options`.

See the `CreateDynaset` method for a description of the possible values of `oradynaset.Options`.

See the `CreateSQL` method for a description of the possible values of `orasqlstmt.Options`

See Also:

- CreateDynaset Method on page 10-85
- CreateSQL Method on page 10-111
- OpenDatabase Method on page 10-212

OraIDataType Property

Applies To

OraField Object on page 9-33

Description

Returns the Oracle internal data type code for the field specified. Not available at design time and read-only at run time.

Usage

```
field_idatatype = orafield.OraIDataType
```

Data Type

`Long Integer`

Remarks

The following Oracle Internal data types are returned.

Constant	Value	Internal Data Type
ORATYPE_VARCHAR2	1	VARCHAR2
ORATYPE_NUMBER	2	NUMBER
ORATYPE_LONG	8	LONG
ORATYPE_DATE	12	DATE
ORATYPE_RAW	23	RAW
ORATYPE_LONGRAW	24	LONG RAW
ORATYPE_CHAR	96	CHAR
ORATYPE_BFLOAT	100	BINARY_FLOAT
ORATYPE_BDOUBLE	101	BINARY_DOUBLE
ORATYPE_MLSLABEL	105	MLSLABEL
ORATYPE_OBJECT	108	OBJECT
ORATYPE_REF	110	REF
ORATYPE_CLOB	112	CLOB
ORATYPE_BLOB	113	BLOB
ORATYPE_BFILE	114	BFILE
ORATYPE_TIMESTAMP	187	TIMESTAMP
ORATYPE_TIMESTAMPTZ	188	TIMESTAMP WITH TIME ZONE
ORATYPE_INTERVALYM	189	INTERVAL YEAR TO MONTH
ORATYPE_INTERVALDS	190	INTERVAL DAY TO SECOND
ORATYPE_TIMESTAMPLTZ	232	TIMESTAMPWITHLOCALTIMEZONE
ORATYPE_VARRAY	247	VARRAY

Constant	Value	Internal Data Type
ORATYPE_TABLE	248	NESTED TABLE

These values can be found in the *ORACLE_BASE**ORACLE_HOME*\oo4o\oraconst.txt file.

See Also:

- OraMaxDSize Property on page 11-117
- OraMaxSize Property on page 11-118
- OraNullOK Property on page 11-119
- OraPrecision Property on page 11-120
- OraScale Property on page 11-121

OraMaxDSize Property

Applies To

OraField Object on page 9-33

Description

Returns the Oracle maximum display size for the field specified. Not available at design time and read-only at run time.

Usage

```
field_maxdisplaysize = orafield.OraMaxDSize
```

Data Type

`Long Integer`

Remarks

This value is meaningful only when the value is returned as a character string, especially when using functions such as `SUBSTR` or `TO_CHAR` to modify the representation of the column.

See Also:

- OraIDataType Property on page 11-115
- OraMaxSize Property on page 11-118
- OraNullOK Property on page 11-119
- OraPrecision Property on page 11-120
- OraScale Property on page 11-121

OraMaxSize Property

Applies To

OraField Object on page 9-33

Description

Returns the Oracle maximum column size as stored in the Oracle data dictionary. Not available at design time and read-only at run time.

Usage

```
field_maxsize = orafield.OraMaxSize
```

Data Type

`Long Integer`

Remarks

The return value is dependent on the Oracle internal data type. The following values will be returned:

Oracle Column Type	Value
`CHAR`, `VARCHAR2`, `RAW`	Length of the column in the table
`NUMBER`	`22` (the internal length)
`DATE`	`7` (the internal length)
`LONG`, `LONG RAW`	`0`
`ROWID`	System dependent
Functions returning internal data type `1`, such as `TO_CHAR()`	Same as `orafield.MaxDSize`

See Also:

- OraIDataType Property on page 11-115
- OraMaxDSize Property on page 11-117
- OraNullOK Property on page 11-119
- OraPrecision Property on page 11-120
- OraScale Property on page 11-121

OraNullOK Property

Applies To

OraField Object on page 9-33

Description

Indicates whether or not `Null` values are permitted for this column. Not available at design time and read-only at run time.

Usage

```
field_nullok = orafield.OraNullOK
```

Data Type

`Integer` (Boolean)

Remarks

This property returns `True` if `Null` values are permitted, otherwise, it returns `False`.

See Also:

- OraIDataType Property on page 11-115
- OraMaxDSize Property on page 11-117
- OraMaxSize Property on page 11-118
- OraPrecision Property on page 11-120
- OraScale Property on page 11-121

OraPrecision Property

Applies To

OraField Object on page 9-33

Description

Returns the precision of a numeric column. Not available at design time and read-only at run time.

Usage

```
field_precision = orafield.OraPrecision
```

Data Type

`Long Integer`

Remarks

This value is meaningful only when the value returned is numeric. Precision is the total number of digits of a number.

See Also:

- OraIDataType Property on page 11-115
- OraMaxDSize Property on page 11-117
- OraMaxSize Property on page 11-118
- OraNullOK Property on page 11-119
- OraScale Property on page 11-121

OraScale Property

Applies To

OraField Object on page 9-33

Description

Returns the scale of a numeric column. Not available at design time and read-only at run time.

Usage

```
field_scale = orafield.OraScale
```

Data Type

`Long Integer`

Remarks

This value is meaningful only when the value returned is numeric. The SQL types `REAL`, `DOUBLE PRECISION`, `FLOAT`, and `FLOAT(`*`N`*`)` return a scale of `-127`.

See Also:

- OraIDataType Property on page 11-115
- OraMaxDSize Property on page 11-117
- OraMaxSize Property on page 11-118
- OraNullOK Property on page 11-119
- OraPrecision Property on page 11-120

Parameters Property

Applies To

OraDatabase Object on page 9-28

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
```

Data Type

OLE Object (`OraParameters`)

Remarks

You can access the parameters in this collection by subscripting (using ordinal integer numbers) or by using the name the parameter that 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 indexes 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.

See Also:

- Add Method on page 10-8
- Count Property on page 11-31
- OraParameter Object on page 9-50
- OraParameters Collection on page 9-68
- Remove Method on page 10-230

PinOption (OraRef) Property

Applies To

OraRef Object on page 9-52

Description

Gets and sets the `Pin` option for the referenceable object during the pin operation.

Usage

```
pin_option = OraRef.PinOption
OraRef.PinOption = pin_option
```

Arguments

Arguments	Description
[in] *PinOption*	An `Integer` representing the `Pin` option.

Data Type

`Integer` (Boolean)

Remarks

Possible values returned by the `pin_option` property are:

Constant	Value	Description
`ORAREF_READ_ANY`	3	If the object is already in the object cache, returns it, otherwise, retrieves it from the database(default).
`ORAREF_READ_RECENT`	4	If the object is retrieved into the cache during a transaction, returns it from the cache, otherwise retrieves the object from the database.
`ORAREF_READ_LATEST`	5	Always retrieves the latest values from the database.

Examples

The following example shows the usage of the `PinOption` property. Before running the sample code, make sure that you have the necessary data types and tables in the database. See "Schema Objects Used in the OraObject and OraRef Examples" on page A-3.

```
Dim OraSession as OraSession
Dim OraDatabase as OraDatabase
Dim OraDynaset as OraDynaset
Dim Person as OraRef

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

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

```
'create a dynaset object from customers
set OraDynaset = OraDatabase.CreateDynaset("select * from customers",  0&)

'retrieve a aperson column from customers. Here Value
'property of OraField object returns Person OraRef
set Person = OraDynaset.Fields("aperson").Value

'set the ORAREF_READ_LATEST read option on the Person object.
Person.PinOption = ORAREF_READ_LATEST

'pin the Person Ref and get the latest copy of referenceable
'object for Ref from the database
MsgBox Person.Name
MsgBox Person.Age
```

PollingAmount Property

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Gets or sets the total amount to be read or written for multiple chunk `Read` and `Write` operations (polling). A value of zero means that polling is not used. This property is read/write at run time.

Usage

```
pollamountbytes = OraBFile.PollingAmount
OraBfile.PollingAmount = pollamountbytes

pollamountbytes = OraBlob.PollingAmount
OraBlob.PollingAmount = pollamountbytes

pollamountchars= OraClob.PollingAmount
OraClob.PollingAmount = pollamountchars
```

Data Type

`Integer`

Remarks

This value is expressed in bytes for the `OraBLOB` and `OraBFILE` objects, or characters for the `OraCLOB` object. It is set before beginning a multiple-chunk read or write operation. After it is set, a series of `Read` or `Write` operations must be issued until the LOB `Status` property no longer returns `ORALOB_NEED_DATA` .This occurs when the `PollingAmount` bytes or characters have been read. Attempting to do other LOB operations before the end of the polling operation results in an error.

Priority (OraAQMsg) Property

Applies To

OraAQMsg Object on page 9-6

Description

Specifies the priority of the message.

Usage

```
Msg.Priority = msg_priority
```

Data Type

```
Integer
```

Remarks

A smaller number indicates higher priority.

Possible Values are:

- Any integer including negative numbers.
- `ORAAQ_NORMAL` (Default): `0`
- `ORAAQ_HIGH`: `-10`
- `ORAAQ_LOW`: `10`

This property can be set while enqueuing and can then be used for priority-based dequeuing.

RDMSVersion Property

Applies To

OraDatabase Object on page 9-28

Description

Returns the database version.

Usage

```
Set Version = oradatabase.RDBMSVersion
```

Data Type

```
String
```

RecordCount Property

Applies To

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

Description

- `OraDynaset`

  Returns the total number of records in the dynaset.

- `OraSQLStmt`

  Returns the number of records processed in an insert, update, or delete statement, even when there is a failure executing the SQL statement.

Not available at design time and read-only at run time.

Usage

```
record_count = oradynaset.RecordCount
record_count = orasqlstmt.RecordCount
```

Data Type

`Long Integer`

Remarks

Referencing this property requires that the entire result table be fetched immediately from an Oracle database to determine the count of records. Due to the potentially severe performance impact of this, the user should avoid using this property and instead execute an additional query using the `COUNT(*)` clause, and use the `SnapshotID` property to guarantee time consistency. For an example, see the `SnapShot` property.

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

Examples

RecordCount Example (OraDynaset)

This example demonstrates the use of the `RecordCount` property to determine the number of records retrieved with a `SELECT` statement and `OraDynaset`. 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 'Create the OraSession Object.
```

```
 Set OraSession = CreateObject("OracleInProcServer.XOraSession")

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

 'Create the dynaset.
 Set OraDynaset = OraDatabase.CreateDynaset("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
```

Record Count Example (OraSQLStmt)

The following example shows the number of records inserted into the database after using an `INSERT` statement with `OraSQLStmt`.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim OraSqlStmt As OraSQLStmt
Dim OraPArray1 As OraParamArray
Dim OraPArray2 As OraParamArray
Dim I As Integer

On Error GoTo ERR_array_sql

'Test case for inserting/updating/deleting multiple rows using parameter arrays
'with SQL statements
Set OraSession = CreateObject("OracleInProcServer.XOraSession")
Set OraDatabase = OraSession.OpenDatabase("exampledb", "scott/tiger", 0&)

'Create table
OraDatabase.ExecuteSQL ("create table part_nos(partno number, description" & _
                  "char(50), primary key(partno))")

OraDatabase.Parameters.AddTable "PARTNO", ORAPARM_INPUT, ORATYPE_NUMBER, 10, 22
OraDatabase.Parameters.AddTable "DESCRIPTION", ORAPARM_INPUT, ORATYPE_CHAR, _
              10, 50
If OraDatabase.LastServerErr <> 0 Or OraDatabase.LastServerErrText <> "" Then
  MsgBox "Error"
End If

Set OraPArray1 = OraDatabase.Parameters("PARTNO")
Set OraPArray2 = OraDatabase.Parameters("DESCRIPTION")

'Initialize arrays
For I = 0 To 9
  achar = "Description" + Str(I)
  OraPArray1.put_Value 1000 + I, I
  OraPArray2.put_Value achar, I
Next I

Set OraSqlStmt = OraDatabase.CreateSql("insert into" & _
          "part_nos(partno, description) values(:PARTNO,:DESCRIPTION)", 0&)
```

```
If OraDatabase.LastServerErr <> 0 Or OraDatabase.LastServerErrText <> "" Then
    MsgBox "Error"
End If
MsgBox "# of records inserted : " & OraSqlStmt.RecordCount

Exit Sub
ERR_array_sql:

    MsgBox Err.Description
```

See Also:

- SnapShot Property on page 11-146
- CreateDynaset Method on page 10-85
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199

RelMsgId (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies that the message of this queue object is enqueued ahead of the message specified by the message ID.

Usage

```
OraAq.RelMsgid = msg_id
```

Data Type

`String`

Remarks

This method is applicable only for an enqueue operation.

Possible values include:

- Any valid message identifier, specified by an array of bytes.
- `ORAAQ_NULL_MSGID` (Default): No message identifier specified.

Setting this property invokes enqueue with the `ORAAQ_ENQ_BEFORE` option. Set this property to `ORAAQ_NULL_MSGID` to place the message on top of the queue.

RowPosition Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the row number of the current row in the dynaset. Not available in design time and read-only in run time.

Usage

```
rownum = OraDynaset.RowPosition
```

Data Type

```
Integer
```

See Also: OraField Object on page 9-33

SafeArray (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Gets or sets the element values from the `Variant SAFEARRAY`.

Usage

```
SafeArray = OraCollection.SafeArray
OraCollection.SafeArray = SafeArray
```

Arguments

Arguments	Description
SafeArray	A `Variant` representing `SafeArray` format.

Data Type

A `Variant` representing a `SafeArray` format.

Remarks

This property is only valid for simple scalar elements types, such as `VARCHAR2` and `NUMBER`. This property raises an error for element type LOBS, Objects, Refs, and so on.

The `Variant SAFEARRAY` index starts at `0`. When converting to `SAFEARRAY` format, the `OraCollection` object converts its element value to its corresponding `SAFEARRAY Variant` type. The following table explains collection element types and their corresponding `SAFEARRAY Variant` types:

Collection Element Type	SAFEARRAY of
Date	String
Number	String
CHAR, VARCHAR2	String
Real	Real
Integer	Integer

For setting a `SAFEARRAY` to a collection, `OraCollection` converts the `SAFEARRAY` elements to its nearest collection element type.

Second (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Second` attribute of an `OraTimeStamp` object.

Usage

```
second = OraTimeStampObj.Second
OraTimeStampObj.Second= second
```

Arguments

Arguments	Description
[in] *second*	The `Second` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Second (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Second` attribute of an `OraTimeStampTZ` object.

Usage

```
second = OraTimeStampTZObj.Second
OraTimeStampTZObj.Second= second
```

Arguments

Arguments	Description
[in] *second*	The `Second` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Seconds Property

Applies To

OraIntervalDS Object on page 9-35

Description

Gets and sets the `Seconds` attribute of an `OraIntervalDS` object.

Usage

```
seconds = OraIntervalDSObj.Seconds
OraIntervalDSObj.Seconds = seconds
```

Arguments

Arguments	Description
[in] *seconds*	An `Integer` specifying the value of the `Seconds` attribute of the `OraIntervalDS` object.

Data Type

`Integer`

Server Property

Applies To

OraDatabase Object on page 9-28

Description

Returns the `OraServer` object to which this object is attached.

Usage

```
Set oraserver = oradatabase.Server
```

Data Type

OLE Object (`OraServer`)

See Also: OraServer Object on page 9-56

ServerType Property

Applies To

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

Description

Specifies the Oracle external type of a SQL or PL/SQL bind variable. Not available at design time and read/write at run time.

Read-only for the `OraParamArray` object. Specify the `ServerType` property during the `AddTable` method.

Usage

```
oraparameter.ServerType = oracle_type
```

Data Type

`Integer`

Remarks

Used to specify the external data type of SQL or PL/SQL (in/out) bind variables. This is necessary because 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.

After an `OraParameter` object has been set to `ServerType` `BLOB`, `CLOB`, `BFILE`, `OBJECT`, `REF`, `VARRAY`, or `NESTED TABLE`, it cannot be changed to any other `ServerType` property.

The following Oracle external data types are supported.

Constant	Value	Internal 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_LONG	8	LONG
ORATYPE_VARCHAR	9	VARCHAR
ORATYPE_DATE	12	DATE
ORATYPE_RAW	23	RAW
ORATYPE_LONGRAW	24	LONG RAW
ORATYPE_UINT	68	UNSIGNED INTEGER
ORATYPE_CHAR	96	CHAR
ORATYPE_CHARZ	97	Null Terminated CHAR

Constant	Value	Internal Data Type
ORATYPE_BFLOAT	100	BINARY_FLOAT
ORATYPE_BDOUBLE	101	BINARY_DOUBLE
ORATYPE_CURSOR	102	PLSQL CURSOR
ORATYPE_MLSLABEL	105	MLSLABEL
ORATYPE_OBJECT	108	OBJECT
ORATYPE_REF	110	REF
ORATYPE_CLOB	112	CLOB
ORATYPE_BLOB	113	BLOB
ORATYPE_BFILE	114	BFILE
ORATYPE_TIMESTAMP	187	TIMESTAMP
ORATYPE_TIMESTAMPTZ	188	TIMESTAMP WITH TIMEZONE
ORATYPE_INTERVALYM	189	INTERVAL YEAR TO MONTH
ORATYPE_INTERVALDS	190	INTERVAL DAY TO SECOND
ORATYPE_TIMESTAMPLTZ	232	TIMESTAMP WITH LOCAL TIME ZONE
ORATYPE_VARRAY	247	VARRAY
ORATYPE_TABLE	248	NESTED TABLE
ORATYPE_RAW_BIN	2000	RAW

These values can be found in the *ORACLE_BASE**ORACLE_HOME*\oo4o\oraconst.txt file.

Examples

This example demonstrates the Add and Remove parameter methods, the ServerType parameter property, and 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

 '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.ExecuteSQL ("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.ExecuteSQL ("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
```

See Also:

- Add Method on page 10-8
- Remove Method on page 10-230
- AddTable Method on page 10-23
- ExecuteSQL Method on page 10-144

Session Property

Applies To

OraCollection Object on page 9-27

OraDatabase Object on page 9-28

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

OraServer Object on page 9-56

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
Set orasession = orasqlstmt.Session
Set orasession = oraserver.Session
```

Data Type

OLE Object (`OraSession`)

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 session, which is, by default, the session associated with the application.

- `oradynaset.Session`

  Returns the `OraSession` object associated with this `OraDynaset` object.

- `orasqlstmt.Session`

  Returns the `OraSession` object associated with this `OraSQLStmt` object.

See Also:

- OraSession Object on page 9-58
- OraSessions Collection on page 9-69

Sessions Property

Applies To

OraClient Object on page 9-18

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
```

Data Type

OLE Object (`OraSessions`)

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 indexes and invalid names return a `Null` `OraSession` object.

See Also:

- Count Property on page 11-31
- OraSession Object on page 9-58
- OraSessions Collection on page 9-69

Size Property

Applies To

OraField Object on page 9-33

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
```

Data Type

`Long Integer`

Remarks

This property returns `0` for `LONG` or `LONG RAW` fields. Use the `FieldSize` method to determine the length of `LONG` or `LONG RAW` fields.

See Also:

- OraFields Collection on page 9-67
- FieldSize Method on page 10-150
- Type Property on page 11-164

Size (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns the current size of the given collection. Read-only at run time.

Usage

```
coll_size = OraCollection.Size
```

Data Type

```
Integer
```

Remarks

For an `OraCollection` object of type `ORATYPE_TABLE`, this property returns the current size of the collection including deleted elements.

See Also: OraField Object on page 9-33

Size (OraLOB and OraBFILE) Property

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Returns the number of bytes in `OraBLOB` and `OraBFILE` objects or the number of characters in an `OraCLOB` object. Read-only.

Usage

```
bytes = OraBFile.Size

bytes = OraBlob.Size

chars = OraClob.Size
```

SnapShot Property

Applies To

OraDynaset Object on page 9-30

Description

Returns the SnapshotID.

Read and write at run time.

Usage

```
SnapshotID = OraDynaset.Snapshot
```

Remarks

The SnapshotID represents the snapshot from which this dynaset was created. It can be thought of as a timestamp. It can be passed into other CreateDynaset method calls to cause them to be created using data from the same point in time as the original dynaset.

The Snapshot property can be set with the value of another Snapshot. That new snapshot is used during the next Refresh operation when the query is reexecuted. The Snapshot property always returns the SnapshotID on which this OraDynaset object was based, not any other SnapshotID set through the snapshot property.

The SnapshotID becomes invalid after a certain amount of time; that amount of time is dependent on the amount of activity and the configuration of the database. When this happens, you get a Snapshot too old error message. For more information about snapshots, see the *Oracle Database Concepts*.

This SnapshotID represents the point in time when this dynaset was created. Changes to this dynaset (Edit, Delete, and AddNew operations) is not reflected in additional dynasets created using this SnapshotID because they occurred after that point in time.

SnapshotID objects are only meaningful for SELECT statements where the tables referenced are real database tables, as opposed to pseudo tables such as DUAL.

One valuable use of the SnapshotID is to calculate the number of rows in a table without using the RecordCount property which causes every row to be fetched. See "Example: Counting Rows in a Dynaset" on page 11-147.

Data Type

Object

Examples

Example: Using the SnapShot Property

This example shows the use of the SnapShot property.

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset1 As OraDynaset
Dim OraDynaset2 As OraDynaset
```

```
Dim SnapshotID as SnapshotID

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

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

'ALLEN's JOB is initially SALESMAN
OraDatabase.ExecuteSql("Update EMP set JOB = 'SALESMAN' where ENAME = 'ALLEN'")

 'Create initial OraDynaset Object.
 Set OraDynaset1 = OraDatabase.CreateDynaset("select empno, ename," & _
            "job from emp", 0&)
MsgBox "OraDynaset1 -- Value of JOB is " & OraDynaset1.Fields("JOB").Value

'Change Allen's JOB
OraDatabase.ExecuteSql("Update EMP set JOB = 'CLERK' where ENAME = 'ALLEN'")

'This SnapshotID represents the point in time in which OraDynaset1 was created
Set SnapshotID = OraDynaset1.Snapshot

'Create OraDynaset2 from the same point in time as OraDynaset1
Set OraDynaset2 = OraDatabase.CreateDynaset("select JOB from EMP" & _
               "where ENAME = 'ALLEN'", 0&, SnapshotID)

MsgBox "OraDynaset2 -- Value of JOB from point of time of OraDynaset1 is " & _
                OraDynaset2.Fields("JOB").Value

'We set the snapshot to NULL which will get us current point in time.
OraDynaset2.Snapshot = Null

'We refresh it and it will get us the data from the current point in time
OraDynaset2.Refresh
MsgBox "OraDynaset2 -- Value of JOB from current point of time is " & _
                OraDynaset2.Fields("JOB").Value

'And back again to the old point in time --
OraDynaset2.Snapshot = SnapshotID
OraDynaset2.Refresh
MsgBox "OraDynaset2 -- Value of JOB from point of time of OraDynaset1 is " & _
              OraDynaset2.Fields("JOB").Value
```

Example: Counting Rows in a Dynaset

This example counts the number of rows in a dynaset without using the `RecordCount` property, which fetches every row. Note that the record count this returns cannot take into account any `AddNew` or `Delete` operations, making the information meaningful only immediately after the dynaset is created

```
Dim OraSession As OraSession
Dim OraDatabase As OraDatabase
Dim OraDynaset As OraDynaset
Dim OraDynCount As OraDynaset
Dim SnapshotID as SnapshotID

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

 'Create the OraDatabase Object by opening a connection to Oracle.
```

```
 Set OraDatabase = OraSession.OpenDatabase("ExampleDb", "scott/tiger", 0&)

'Create the Dynaset
Set OraDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)
Set SnapshotID = OraDynaset.Snapshot

'Use the snapshot for count query to guarantee the same point in time
Set OraDynCount = OraDatabase.CreateDynaset("select count(*) NUMROWS" & _
         "from emp", 0&, SnapshotID)
MsgBox "Number of rows in the table is " & OraDynCount.Fields("NUMROWS").Value
```

See Also:

- CreateDynaset Method on page 10-85
- CreateCustomDynaset Method on page 10-80
- *Oracle Database Concepts*

Sort Property

Remarks

The `OraDynaset` object does not support this property. Sort your record set by using a SQL `ORDER BY` clause.

See Also:

- CreateDynaset Method on page 10-85
- OraDynaset Object on page 9-30

SQL Property

Applies To

OraDynaset Object on page 9-30

OraSQLStmt Object on page 9-60

Description

Returns or sets the SQL statement used to create the specified dynaset or `OraSQLStmt` object. Not available at design time and read/write at run time.

Usage

```
SQL_statement  = oradynaset.SQL
SQL_statement  = orasqlstmt.SQL

oradynaset.SQL = SQL_statement
orasqlstmt.SQL = SQL_statement
```

Data Type

`String`

Remarks

The first use returns the contents of the SQL statement buffer, and the second use sets the contents of the SQL statement buffer.

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

Examples

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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

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

 'Create the OraDynaset Object.
 Set OraDynaset = OraDatabase.CreateDynaset("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.Refresh

'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
```

See Also: Refresh Method on page 10-225

Status Property

Applies To

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

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
parameter_status = oraparamarray.Status
```

Data Type

`Integer`

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 auto-enabled.

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 *ORACLE_BASE**ORACLE_HOME*\oo4o\oraconst.txt file.

Examples

This example demonstrates the use of parameters and the `ExecuteSQL` method 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

```
 'Create the OraDatabase Object by opening a connection to Oracle.
 Set OraDatabase = OraSession.OpenDatabase("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.ExecuteSQL ("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
```

See Also:

- Add Method on page 10-8
- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41
- ExecuteSQL Method on page 10-144
- Remove Method on page 10-230

Status (OraLOB/BFILE) Property

Applies To

OraBLOB, OraCLOB Objects on page 9-11

OraBFILE Object on page 9-9

Description

Returns the status of the current polling operation.

Read-only.

Usage

```
status = OraBFile.Status
status  = OraBlob.Status
status = OraClob.Status
```

Data Type

`Integer`

Remarks

This value only has meaning when the `PollingAmount` property is not zero, and a `Read` operation has occurred. Possible return values are:

- `ORALOB_NEED_DATA`

  There is more data to be read or written.

- `ORALOB_NODATA`

  There is no data to be read or written, usually due to an error condition.

- `ORALOB_SUCCESS` LOB

  The data was read or written successfully.

See Also: PollingAmount Property on page 11-125

Subscriptions Property

Applies To

OraDatabase Object on page 9-28

Description

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

Usage

```
Set orasubscriptions_collection = oradatabase.Subscriptions
```

Data Type

OLE Object (`OraSubscriptions`)

Remarks

You can access the subscriptions in this collection by subscripting (using ordinal integer numbers). You can obtain the number of subscriptions in the collection using the `Count` property of the returned collection. Integer subscripts begin with 0 and end with `Count-1`. Out-of-range indexes return a `Null` `OraSubscription` object.

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

Examples

See "Example: Registering an Application for Notification of Database Events" on page 10-15 for a complete example.

See Also:

- "Database Events" on page 4-22
- OraSubscription Object on page 9-61
- OraSubscriptions Collection on page 9-70
- Remove (OraSubscriptions Collection) Method on page 10-231

TableName (OraRef) Property

Applies To

OraRef Object on page 9-52

Description

A `String` containing the name of the object table in which the underlying referenceable object resides.

Usage

```
table_name = OraRef.TableName
```

Data Type

`String`

Remarks

This property is read-only.

TableSize (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns the current size of the given collection. Read-only at run time.

Usage

```
table_size = OraCollection.TableSize
```

Data Type

`Integer`

Remarks

For an `OraCollection` object of type `ORATYPE_TABLE`, it returns the current size of the collection, excluding deleted elements.

TimeZone (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the time zone information of an `OraTimeStampTZ` object.

Usage

```
timezone = OraTimeStampTZObj.TimeZone
OraTimeStampTZObj.TimeZone= timezone
```

Arguments

Arguments	Description
[in] *timezone*	The time zone attribute of an `OraTimeStampTZ` object.

Data Type

`String`

Remarks

Setting the `TimeZone` property does not change the Coordinated Universal Time (**UTC**) datetime values stored in the `OraTimeStampTZ` object. However, the local datetime values in the specified time zone can change.

The following table shows the UTC datetime values that correspond to the datetime and time zone values of the `OraTimeStampTZ` object in the example.

Properties	OraTSTZ Object Values	UTC Date Time Values of the OraTSTZ Object
`Year`	`2003`	`2003`
`Month`	`4`	`4`
`Day`	`29`	`29`
`Hour`	`12`	`19`
`Minute, Second, Nanosecond`	`0`	`0`
`TimeZone`	`-07:00`	`00:00`

Setting the `TimeZone` property to `-08:00` changes the datetime values in the specified time zone of the `OraTimeStampTZ` object, but does not change the UTC datetime values.

Properties	New OraTSTZ Object Values	UTC Date Time Values of the New OraTSTZ Object
`Year`	`2003`	`2003`
`Month`	`4`	`4`

Properties	New OraTSTZ Object Values	UTC Date Time Values of the New OraTSTZ Object
Day	29	29
Hour	11	19
Minute, Second, Nanosecond	0	0
TimeZone	-08:00	00:00

Examples

```
Dim OraTSTZ as OraTimeStampTZ
Dim OraTSTZ_new as OraTimeStampTZ
Dim OraTSTZStr as String
Dim OraTSTZStr_new as String
Set OraTSTZ = oo4oSession.CreateOraTimeStampTZ( "2003-APR-29" & _
       "12:00:00 -07:00", "YYYY-MON-DD HH:MI:SS TZH:TZM")

'Change Time Zone to "-08:00"
Set OraTSTZ_new = OraTSTZ.Clone
oraTSTZ_new.TimeZone = "-08:00"

'OraTSTZStr has value as (29-APR-03 12.00.00.000000000 PM -07:00)
OraTSTZStr = OraTSTZ.value
'OraTSTZStr_new has value as (29-APR-03 11.00.00.000000000 PM -08:00)
OraTSTZStr_new = OraTSTZ_new.value
```

TotalDays Property

Applies To

OraIntervalDS Object on page 9-35

Description

Gets and sets the total number of days that this `OraIntervalDS` object represents.

Usage

```
totalDays = OraIntervalDSObj.TotalDays
OraIntervalDSObj.TotalDays = totalDays
```

Arguments

Arguments	Description
[in] *totalDays*	A `Variant` type of any numeric value or an `OraNumber` object specifying the `OraIntervalDS` object as the total number of days.

Data Type

`Double`

Examples

```
Dim oraIDS   as OraIntervalDS
'Create an OraIntervalDS using a string which represents 1 day and 12 hours
Set oraIDS = oo4oSession.CreateOraIntervalDS("1 12:0:0.0")

'totalDays is set to 1.5 which represents an interval of 1.5 days
totalDays = oraIDS.TotalDays
```

See Also: OraNumber Object on page 9-41

TotalYears Property

Applies To

OraIntervalYM Object on page 9-37

Description

Gets and sets the total number of years that this `OraIntervalYM` object represents.

Usage

```
totalYears = OraIntervalYMObj.TotalYears
OraIntervalYMObj.TotalYears= totalYears
```

Arguments

Arguments	Description
[in] *totalYears*	A `Variant` type of any numeric value specifying the `OraIntervalYM` object as the total number of years.

Data Type

`Double`

Examples

```
Dim oraIYM   as OraIntervalYM

'Create an OraIntervalYM using a string which represents 1 year and 6 months
Set oraIYM = oo4oSession.CreateOraIntervalYM("1-6")
'totalYears is set to 1.5 which represents an interval of 1.5 years
totalYears = oraIYM.TotalYears
```

Transactions Property

Applies To

OraDynaset Object on page 9-30

Description

Indicates 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
```

Data Type

`Integer` (Boolean)

Remarks

This property always returns `True`.

See Also:

- BeginTrans Method on page 10-43
- CommitTrans Method on page 10-66
- OraDynaset Object on page 9-30
- ResetTrans Method on page 10-233
- Rollback Method on page 10-235

Truncated Property

Applies To

OraField Object on page 9-33

Description

Indicates 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
```

Data Type

`Integer` (Boolean)

Remarks

This property returns `True` if truncated data is returned; otherwise, it returns `False`. Truncation can only occur for `LONG` or `LONG RAW` fields. Use this property to decide whether more data needs to be retrieved from an Oracle database using the `GetChunk` method.

See Also:

- GetChunk Method on page 10-156
- Type Property on page 11-164

Type Property

Applies To

OraField Object on page 9-33

OraParameter Object on page 9-50

OraParamArray Object on page 9-47

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
data_type = oraparamarray.Type
```

Data Type

`Integer`

Remarks

- `orafield.Type`

  Returns the `Variant` data type (see Visual Basic documentation) associated with the returned value of this field.

- `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 Oracle Database.

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

Oracle Data Type	Constant	Value	Data Type
BINARY_DOUBLE	ORADB_DOUBLE	7	Double
BINARY_FLOAT	ORADB_SINGLE	6	Single
BLOB	ORADB_OBJECT	9	OraBLOB
CHAR	ORADB_TEXT	10	String
CLOB	ORADB_OBJECT	9	OraCLOB
DATE	ORADB_DATE	8	Variant
DATE	ORADB_DATE	8	Date
INTERVAL DAY TO SECOND	ORADB_OBJECT	9	OraIntervalDS
INTERVAL YEAR TO MONTH	ORADB_OBJECT	9	OraIntervalYM
LONG	ORADB_MEMO	12	String
LONG RAW	ORADB_LONGBINARY	11	String

Oracle Data Type	Constant	Value	Data Type
NESTED TABLE	ORADB_OBJECT	9	OraBFILE
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	String
NUMBER (1-15, n)	ORADB_DOUBLE	7	Double
NUMBER (16-38, n)	ORADB_TEXT	10	String
RAW	ORADB_LONGBINARY	11	String
REF	ORADB_OBJECT	9	OraCollection
TIMESTAMP	ORADB_OBJECT	9	OraTimeStamp
TIMESTAMP WITH LOCAL TIME ZONE	ORADB_OBJECT	9	OraTimeStamp
TIMESTAMP WITH TIME ZONE	ORADB_OBJECT	9	OraTimeStampTZ
VARRAY	ORADB_OBJECT	9	OraCollection
VARCHAR2	ORADB_TEXT	10	String

These values are located in the *ORACLE_BASE**ORACLE_HOME*\oo4o\oraconst.txt file and are intended to match similar constants in the Visual Basic file datacons.txt file.

Note that fields of type DATE are returned in the default Visual Basic format as specified in the Control Panel, even though the default Oracle date format is "DD-MMM-YY".

Note that columns defined as NUMBER instead of NUMBER(precision,scale) are, by definition, floating point numbers with a precision of 38. This means that the Type property returns a type of ORADB_TEXT for these columns.

See Also: Value Property on page 11-173

Type (OraAttribute) Property

Applies To

OraAttribute Object on page 9-7

Description

A integer code representing the type of this attribute.

Usage

```
typecode = OraAttribute.Type
```

Data Type

```
Integer
```

Remarks

These integer codes correspond to external data types in Oracle Call Interface (OCI). See Oracle data types.

See Also: "Oracle Data Types" on page A-1

Type (OraCollection) Property

Applies To

OraCollection Object on page 9-19

Description

Returns the type code of the collection.

Usage

```
coll_type = OraCollection.Type
```

Data Type

```
Integer
```

Remarks

This property returns one of the following values:

Constant	Value	Description
ORATYPE_VARRAY	247	Collection is VARRAY type.
ORATYPE_TABLE	248	Collection is nested table type.

Type (OraMetaData) Property

Applies To

OraMetaData Object on page 9-39

Description

Returns type of the schema object described by the `OraMetaData` object.

Usage

```
type = OraMetaData.Type
```

Remarks

The possible values include the following constants:

Constants	Value
ORAMD_TABLE	1
ORAMD_VIEW	2
ORAMD_COLUMN	3
ORAMD_COLUMN_LIST	4
ORAMD_TYPE	5
ORAMD_TYPE_ATTR	6
ORAMD_TYPE_ATTR_LIST	7
ORAMD_TYPE_METHOD	8
ORAMD_TYPE_METHOD_LIST	9
ORAMD_TYPE_ARG	10
ORAMD_TYPE_RESULT	11
ORAMD_PROC	12
ORAMD_FUNC	13
ORAMD_ARG	14
ORAMD_ARG_LIST	15
ORAMD_PACKAGE	16
ORAMD_SUBPROG_LIST	17
ORAMD_COLLECTION	18
ORAMD_SYNONYM	19
ORAMD_SEQENCE	20
ORAMD_SCHEMA	21
ORAMD_OBJECT_LIST	22
ORAMD_OBJECT_LIST	23
ORAMD_DATABASE	24

Note: If this version of the `OraMetaData` object is used on Oracle Database release 8.1 or later, values higher than 24 are possible if the database is enhanced to introduce new schema types.

See Also: ORAMD_TABLE Attributes on page 9-39

TypeName (OraObject and OraRef) Property

Applies To

OraObject Object on page 9-43

OraRef Object on page 9-52

Description

Specifies a `String` containing the name of the user-defined type of the object.

Usage

```
typename = OraRef.TypeName
typename = OraObject.TypeName
```

Data Type

`String`

Remarks

This property is read-only at run time.

Updatable Property

Applies To

OraDynaset Object on page 9-30

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
```

Data Type

`Integer` (Boolean)

Remarks

Returns `True` if the rows in the specified dynaset can be updated; otherwise, it returns `False`.

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.

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.
3. Oracle Database 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`.

Examples

This example demonstrates the use of the `Updatable` method. 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 OraSession
 Dim OraDatabase As OraDatabase
 Dim OraDynaset As OraDynaset

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

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

 'Create an updatable dynaset using a simple query.
```

```
 Set OraDynaset = OraDatabase.CreateDynaset("select * from emp", 0&)

 Call IsDynUpdatable(OraDynaset)

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

 'Create a non-updatable dynaset using a join.
 Set OraDynaset = OraDatabase.CreateDynaset("select ename, emp.deptno," & _
           "loc from emp, dept where emp.deptno = dept.deptno", 0&)
 Call IsDynUpdatable(OraDynaset)

End Sub

Sub IsDynUpdatable (odyn As OraDynaset)

 '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
```

See Also:

- CreateDynaset Method on page 10-85
- SQL Property on page 11-150
- RecordSource Property of Data Control on page 14-31

Value Property

Applies To

OraField Object on page 9-33

OraParameter Object on page 9-50

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
```

Data Type

`Variant`

Remarks

- `Orafield.Value`

  Returns the value of the field as a `Variant`.

  `data_value = orafield.Value` sets the contents of the field. 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 64 KB. You must retrieve data fields larger than 64 KB 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".

The `Value` argument can be an Oracle Database 10*g* object, such as an `OraBLOB`.

Similar to a dynaset, the object obtained from parameter `Value` property always refers to the latest value of the `Parameter`. The Visual Basic value `Null` can also be passed as a value. The Visual Basic value `EMPTY` can be used for `BLOB` and `CLOB` to mean an

empty LOB, and for `OBJECT`, `VARRAY`, and `NESTED TABLE` to mean an object whose attributes are all `Null`.

See Also:

- GetChunk Method on page 10-156
- OraParamArray Object on page 9-47
- Type Property on page 11-164

Value (OraAttribute) Property

Applies To

OraAttribute Object on page 9-7

Description

Gets or sets the value of the attribute. This value could be an instance of an `OraObject`, `OraRef`, or `OraCollection` object, or any of the supported scalar types, such as `Integer` or `Float`.

Usage

```
attr_value = OraAttribute.Value
OraAttribute.Value = attr_value
```

Data Type

`Variant`

Remarks

This is the default property for this object.

The `Value` property of the `OraAttribute` object returns the value of the attribute as a `Variant`. The `Variant` type of the attribute depends on the attribute type of the attribute. Attribute values can be `Null` and can be set to `Null`. For attribute of type objects, `REF`, LOB and Collection, attribute values are returned as corresponding OO4O objects for that type.

The following table identifies the attribute type and the return value of the `Value` property of the `OraAttribute` object:

Element Type	Element Value
`Object`	`OraObject`
`REF`	`OraRef`
`VARRAY`, Nested Table	`OraCollection`
`BLOB`	`OraBLOB`
`CLOB`	`OraCLOB`
`BFILE`	`OraBFILE`
`Date`	`String`
`Number`	`String`
`CHAR,VARCHAR2`	`String`
`Real`	`Real`
`Integer`	`Integer`

Value (OraAQMsg) Property

Applies To

OraAQMsg Object on page 9-6

Description

Returns or sets the value of the given object.

Usage

```
Msg.Value = my_string
set Msg.Value = OraObj

my_string = Msg.Value
Set OraObj = Msg.Value
```

Data Type

`String`

Remarks

The `Value` property represents the actual message for `RAW` as well as user-defined types.

This property is not available at design time and read/write at run time.

Examples

```
'To set the value for a message of Raw type
OraAQMsg.Value = "This is a test message"
myString = "Another way of setting the message"
OraAQMsg.Value = myString

'To set the value for a message of user-defined type
Dim OraObj as OraObject
OraObj("subject").Value = txtdesc
OraObj("text").Value = txtmsg
set OraAQMsg.Value = OraObj

'To get the value from a message of raw type
myString = OraAQMsg.Value

'To get the value from a message of object type(user-defined type)
Set OraObj = OraMsg.Value
txtdesc = OraObj("subject").Value
txtmsg = OraObj("text").Value
```

Value (OraIntervalDS) Property

Applies To

OraIntervalDS Object on page 9-35

Description

When read, the `Value` property provides a string representation of the value of the `OraIntervalDS` object using the format [+/-]Day HH:MI:SSxFF. When set, the `Value` property accepts a `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object.

Usage

```
string = OraIntervalDSObj.Value
OraIntervalDSObj.Value = value
```

Arguments

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalDS` object.

Data Type

`Variant`

Remarks

If the value set is a `Variant` of type `String`, it must be in the following format: [+/-] Day HH:MI:SSxFF.

If the value set is a numeric value, the value provided should represent the total number of days that the `OraIntervalDS` object represents.

Examples

```
Dim oraIDS as OraIntervalDS

'Create an OraIntervalDS using a string which represents 1 day and 12 hours
Set oraIDS = oo4oSession.CreateOraIntervalDS("1 12:0:0.0")

'get the OraIntervalDS.Value return a string for the Value
' property, idsStr is set to "01 12:00:00.000000"
idsStr = oraIDS.Value

'can also return a string for the Value property as follows
idsStr = oraIDS

'set the OraIntervalDS.Value using a string which represents 1 days and 12 hours
oraIDS.Value = "1 12:0:0.0"

'set the OraIntervalDS.Value using a numeric value which represents
'1 days and 12 hours
oraIDS.Value = 1.5
```

See Also: CreateOraIntervalDS Method on page 10-92

Value (OraIntervalYM) Property

Applies To

OraIntervalYM Object on page 9-37

Description

When read, the `Value` property provides a string representation of the value of the `OraIntervalYM` object using the format YEARS-MONTHS.

When set, the `Value` property accepts a `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object.

Usage

```
string = OraIntervalYMObj.Value
OraIntervalYMObj.Value= value
```

Arguments

Arguments	Description
[in] *value*	A `Variant` of type `String`, a numeric value, or an `OraIntervalYM` object.

Data Type

`String`

Remarks

If the value set is a `Variant` of type `String`, it must be in following format: [+/-] YEARS-MONTHS.

If the value set is a numeric value, the value provided should represent the total number of years that the `OraIntervalYM` object represents.

Examples

```
Dim oraIYM as OraIntervalYM

'Create an OraIntervalYM using a string which represents 1 year and 6 months
Set oraIYM = oo4oSession.CreateOraIntervalYM("1-6")

'get the OraIntervalYM.Value return a string for the Value property,
' iymStr is set to "01-06"
iymStr = oraIYM.Value

'can also return a string for the Value property as follows
iymStr = oraIYM

'set the OraIntervalDS.Value using a string which represents 1 year and 6 months
oraIYM.Value = "1-6"

'set the OraIntervalYM.Value using a numeric value which represents
'1 years and 6 months
oraIYM.Value = 1.5
```

See Also: CreateOraIntervalYM Method on page 10-94

Value (OraMDAttribute) Property

Applies To

OraMDAttribute Object on page 9-38

Description

A String containing the value of the attribute.

Usage

```
value = OraMDAttribute.Value
```

Data Type

`String`

Remarks

This is the default property.

Value (OraNumber) Property

Applies To

OraNumber Object on page 9-41

Description

When read, the `Value` property provides a string representation of the value of the `OraNumber` object using the current format string. When set, the `Value` property accepts a `Variant` of type `String`, `OraNumber`, or a numeric value. Read and write at run time.

Usage

```
string = OraNumber.Value
OraNumber.Value = variantval
```

Arguments

Arguments	Description
[in] *variantval*	A `Variant` of type `String`, `OraNumber`, or a numeric value.

Data Type

`Variant`

Remarks

If the `Value` property is set to a numeric type, such as a `LONG`, it is limited to the maximum precision Visual Basic provides for numerical values.

If the current format cannot be applied successfully to the value, an error is raised. An error is also raised if this property is set to a `Variant` value that cannot be converted to a number, such as a string of nonnumeric characters.

Value (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

When read, the Value property provides a string representation of the value of the OraTimeStamp object. If the Format property is not null, the output string format is in the format specified by the Format property; otherwise, the output string format is in the session TIMESTAMP format (NLS_TIMESTAMP_FORMAT). When set, the Value property accepts a Variant of type String, Date, or OraTimeStamp.

Usage

```
string = OraTimeStampObj.Value
OraTimeStampObj.Value= value
```

Arguments

Arguments	Description
[in] *value*	A Variant of type String, Date, or OraTimeStamp.

Data Type

String

Remarks

If the value is of type String and Format is not null, the string format must match the Format property. If the Format property is null, the string format must match the session TIMESTAMP format.

Examples

```
...

Set OraTimeStamp = OraSession.CreateOraTimeStamp("1999-APR-29 " & _
         "12:10:23.444 AM", "YYYY-MON-DD HH:MI:SS.FF AM")

'returns a string for the Value property
tsStr = OraTimeStamp.Value

'set OraTimeStamp.Value using a string
OraTimeStamp.Value = "1999-APR-29 12:10:23.444 AM"
```

See Also: CreateOraTimeStamp Method on page 10-100

Value (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

When read, the Value property provides a string representation of the value of the OraTimeStampTZ object. If the Format property is not null, the output string format is in the format specified by the Format property; otherwise, the output string format is in the session TIMESTAMP WITH TIME ZONE format (NLS_TIMESTAMP_TZ_FORMAT). When set, the Value property accepts a Variant of type String, Date, or OraTimeStampTZ.

Usage

```
string = OraTimeStampTZObj.Value
OraTimeStampObjTZ.Value= value
```

Arguments

Arguments	Description
[in] *value*	A Variant of type String, Date, or OraTimeStampTZ.

Data Type

String

Remarks

If the Variant is of type String and the Format property is not null, the string format must match the Format property. If the Format property is null, the string format must match the session TIMESTAMP WITH TIME ZONE format.

If the Variant is of type Date, the date-time value in Date is interpreted as the date-time value in the session time zone. The time zone information in the OraTimeStampTZ object contains the session time zone.

Examples

```
Dim OraTimeStampTZ As OraTimeStampTZ
...

Set OraTimeStampTZ = OraSession.CreateOraTimeStampTZ("2003-APR-29" & _
            "12:00:00 -07:00", "YYYY-MON-DD HH:MI:SS TZH:TZM")

'returns a string for the Value property
tstzStr = OraTimeStampTZ.Value
...

'set OraTimeStampTZ.Value using a string
OraTimeStampTZ.Value = "2003-APR-29 12:00:00 -07:00"
```

See Also: CreateOraTimeStampTZ Method on page 10-102

Version (OraObject and Ref) Property

Applies To

OraObject Object on page 9-43

OraRef Object on page 9-52

Description

Returns a String containing user-assigned version of the type of underlying value instance.

Usage

```
version  = OraRef.Version
version  = OraObject.Version
```

Data Type

```
String
```

Remarks

This property is read-only at run time.

Visible (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies the transactional behavior of the enqueue request.

Usage

```
Q.Visible = transaction_mode
```

Data Type

```
Integer
```

Remarks

This property is applicable only for an enqueue operation.

Possible values are:

- `ORAAQ_ENQ_IMMEDIATE` (1)

  The enqueue operation constitutes a transaction of its own. Set this property to make the message visible immediately after the enqueue operation.

- `ORAAQ_ENQ_ON_COMMIT` (2) (Default)

  The enqueue is part of the current transaction, and the message is visible only after the transaction commits.

Examples

```
Msg.Value = "The visibility option used in the enqueue call is " & _
                    "ORAAQ_ENQ_IMMEDIATE"
Q.Visible = ORAAQ_ENQ_IMMEDIATE
Q.Enqueue
```

Wait (OraAQ) Property

Applies To

OraAQ Object on page 9-3

Description

Specifies the wait time (in seconds), if there is currently no message available.

Usage

```
Q.Wait = seconds
```

Data Type

`Integer`

Remarks

Applicable only for a dequeue operation.

Possible values are:

- `ORAAQ_DQ_WAIT_FOREVER` (`-1`) (Default)

  Waits forever.

- `ORAAQ_DQ_NOWAIT` (`0`)

  Does not wait.

XMLAsAttribute Property

Applies To

OraField Object on page 9-33

Description

Gets and sets a Boolean value that indicates whether this field name is given as an attribute. If the value is `False`, the field name is given as an element. Readable and writable at run time.

Usage

```
OraField.XMLAsAttribute = True
```

Remarks

The default value for this property is `False`.

Fields of type `BLOB`, `CLOB`, `BFILE`, `Object`, `VARRAY`, `Nested Table`, `Long` or `LongRaw` cannot be XML attributes.

XMLCollID Property

Applies To

OraDynaset Object on page 9-30

Description

Gets and sets the attribute name that replaces `id` (as in `<TYPENAME_ITEM id = "1">`) in the rendering of collection items that occurs when `GetXML` or `GetXMLToFile` methods are called. Readable and writable at run time.

Usage

```
oradynaset.XMLCollID = "NEWID"
```

Remarks

The default value for this property is `id`. If this property is set to `Null` or an empty `String` (`""`), the `collectionid` attribute is omitted. The attribute name must be valid or an error is raised. The case is preserved.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

XMLEncodingTag Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets a string value in the encoding tag of the generated XML document.

Usage

```
OraDynaset.XMLEncodingTag = "SHIFT_JIS"
```

Remarks

This property is useful when the XML document generated by OO4O is converted to a different character set encoding before it is stored or parsed. This might occur if the property is to be loaded into a database or stored in a file system.

This property only sets the encoding tag value; it does not change the actual encoding of the document. The document generated by the `GetXML` method in Visual Basic is encoded in UCS2. The documents generated by the `GetXMLToFile` method use the same character set as the current `NLS_LANG` setting.

If this property is set to an empty String, the default encoding tags are used. To omit the tag entirely, use `OraDynaset.XMLOmitEncodingTag`.

No validity checking of the chosen encoding is done.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164
- XMLOmitEncodingTag Property on page 11-192

XMLNullIndicator Property

Applies To

OraDynaset Object on page 9-30

Description

Gets and sets a Boolean value that indicates whether a null indicator attribute is used in the case of `Null` field values. If the property is `False`, tags with `Null` values are omitted. Readable and writable at run time.

Usage

```
oradynaset.XMLNullIndicator = True
```

Remarks

The default value for this property is `False`.

XMLOmitEncodingTag Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets a Boolean value that determines if the encoding tag should be omitted.

Usage

```
OraDynaset.XMLOmitEncodingTag = True
```

Remarks

The default value is `False`.

If this property is set to `False`, the value of the `XMLEncodingTag` property is used in the encoding tag.

See Also: XMLEncodingTag Property on page 11-190

XMLRowID Property

Applies To

OraDynaset Object on page 9-30

Description

Gets and sets the attribute name that replaces `id` (as in `<ROW id= "1">`) in the rendering of XML that occurs when `GetXML` or `GetXMLToFile` methods are called. Readable and writable at run time.

Usage

```
oradynaset.XMLRowID = "NEWID"
```

Remarks

The default value for this property is `id`. If this property is set to `Null` or an empty string (`""`), the `rowid` attribute is omitted. The attribute name must be valid or an error is raised. The case is preserved.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

XMLRowsetTag Property

Applies To

OraDynaset Object on page 9-30

Description

Gets or sets the tag name that replaces the rowset tag <ROWSET> in the rendering of XML that occurs when GetXML or GetXMLToFile methods are called. Readable and writable at run time.

Usage

```
oradynaset.XMLRowSetTag = "NEWROWSET"
```

Remarks

The default value for this property is ROWSET. The tag name must be valid or an error is raised. The case is preserved. This tag is the root, unless schema metadata is requested with the document.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

XMLRowTag Property

Applies To

OraDynaset Object on page 9-30

Description

Gets and sets the tag name that replaces <ROW> in the rendering of XML that occurs when GetXML or GetXMLToFile methods are called. Readable and writable at run time.

Usage

```
oradynaset.XMLRowTag = "NEWROW"
```

Remarks

The default value for this property is ROW. If this property is set to Null or an empty string (""), the <ROW> tag is omitted. The tag name must be valid or an error is raised. The case is preserved.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

XMLTagName Property

Applies To

OraField Object on page 9-33

Description

Gets and sets the tag name that is used for this field in the rendering of XML that occurs when `GetXML` or `GetXMLToFile` methods are called. Readable and writable at run time.

Usage

```
orafield.XMLTagName = "EmployeeName"
```

Remarks

The default value for this property is the value of the `Name` property. If this property is set to `Null` or an empty string (`""`), this field is omitted. The name must be valid or an error is raised. The case is preserved.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

XMLUpperCase Property

Applies To

OraDynaset Object on page 9-30

Description

Gets and sets a Boolean value that indicates whether tag and attribute names are uppercase when `GetXML` or `GetXMLToFile` methods are called. Readable and writable at run time.

Usage

```
oradynaset.XMLUpperCase = True
```

Remarks

The default value for this property is `False`. If this property is set to `True`, all of the tag and attribute names are in upper case. This method should be called only *after* all custom tag or attribute names have been set by the user.

See Also:

- GetXML Method on page 10-163
- GetXMLToFile Method on page 10-164

Year (OraTimeStamp) Property

Applies To

OraTimeStamp Object on page 9-62

Description

Returns or sets the `Year` attribute of an `OraTimeStamp` object.

Usage

```
year = OraTimeStampObj.Year
OraTimeStampObj.Year = year
```

Arguments

Arguments	Description
[in] *year*	The `Year` attribute of an `OraTimeStamp` object.

Data Type

`Integer`

Year (OraTimeStampTZ) Property

Applies To

OraTimeStampTZ Object on page 9-64

Description

Returns or sets the `Year` attribute of an `OraTimeStampTZ` object.

Usage

```
year = OraTimeStampObjTZ.Year
OraTimeStampObjTZ.Year = year
```

Arguments

Arguments	Description
[in] *year*	The `Year` attribute of an `OraTimeStampTZ` object.

Data Type

`Integer`

Years Property

Applies To

OraIntervalYM Object on page 9-37

Description

Gets and sets the `Years` attribute of an `OraIntervalYM` object.

Usage

```
years = OraIntervalYMObj.Years
OraIntervalYMObj.Years = years
```

Arguments

Arguments	Description
[in] *years*	An `Integer` specifying the value of the `Years` attribute of the `OraIntervalYM` object.

Data Type

`Integer`

12

Data Control Events

This chapter describes Oracle Data Control Events. For an introduction to Data Control, see "Oracle Data Control" on page 1-4.

> **See Also:** For more information, see the Microsoft Visual Basic help and documentation.

This chapter contains these topics:

DragDrop Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- Drag Method on page 13-2
- DragIcon Property on page 14-13
- DragMode Property on page 14-14
- DragOver Event on page 12-3
- MouseDown Event on page 12-5
- MouseMove Event on page 12-6
- MouseUp Event on page 12-7

DragOver Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- Drag Method on page 13-2
- DragDrop Event on page 12-2
- DragIcon Property on page 14-13
- DragMode Property on page 14-14
- MouseDown Event on page 12-5
- MouseMove Event on page 12-6
- MouseUp Event on page 12-7

Error Event

Applies To

Oracle Data Control on page 1-4

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 run-time error.

See Also:

- AddNew Method on page 10-19
- Delete Method on page 10-116
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199

MouseDown Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- MouseMove Event on page 12-6
- MousePointer Property on page 14-22
- MouseUp Event on page 12-7

MouseMove Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- MouseDown Event on page 12-5
- MousePointer Property on page 14-22
- MouseUp Event on page 12-7

MouseUp Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- MouseDown Event on page 12-5
- MousePointer Property on page 14-22
- MouseMove Event on page 12-6

Reposition Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- Error Event on page 12-4
- FindFirst, FindLast, FindNext, and FindPrevious Methods on page 10-151
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- Validate Event on page 12-9

Validate Event

Applies To

Oracle Data Control on page 1-4

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 Also:

- AddNew Method on page 10-21
- Bookmark Property on page 11-13
- Close Method on page 10-63
- Delete Method on page 10-116
- Edit Method on page 10-134
- EditMode Property on page 11-51
- FindFirst, FindLast, FindNext, and FindPrevious Methods on page 10-151
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- UpdateRecord Method on page 13-6

13

Data Control Methods

This chapter describes Oracle Data Control methods. For an introduction to Data Control, see "Oracle Data Control" on page 1-4.

> **See Also:** For more information, see the Microsoft Visual Basic help and documentation.

This chapter contains these topics:

Drag Method

Applies To

Oracle Data Control on page 1-4

Description

Begins, ends, or cancels dragging controls.

See Also:

- DragDrop Event on page 12-2
- DragIcon Property on page 14-13
- DragMode Property on page 14-14
- DragOver Event on page 12-3
- MousePointer Property on page 14-22

Move Method

Applies To

Oracle Data Control on page 1-4

Description

Moves a form or control.

See Also:

- Height Property on page 14-19
- Left Property on page 14-21
- Top Property on page 14-35
- Width Property on page 14-38

Refresh Method

Applies To

Oracle Data Control on page 1-4

Description

This method recreates 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 on page 14-9
- Database Property on page 14-10
- OraDatabase Object on page 9-28
- OraDynaset Object on page 9-30
- RecordSource Property on page 14-31
- SQL Property on page 11-150

UpdateControls Method

Applies To

Recordset Property on page 14-29 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
```

Example

NOTE: This code snippet is intended to be placed in a complete application. The code snippet cancels changes made to bound controls and restores the data to the original values. To use this code snippet, copy it 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`.

```
Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)

Const KEY_ESCAPE = &H1B
If KeyCode = KEY_ESCAPE Then
oradata1.recordset.UpdateControls
End If
End Sub
```

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: For backward compatibility with earlier .VBX control, this method is also available as the method of data control's Recordset.

See Also:

- Recordset Property on page 14-29
- UpdateRecord Method on page 13-6

UpdateRecord Method

Applies To

Recordset Property on page 14-29 of the Oracle Data Control.

Description

Saves the current values of bound controls.

```
oradata1.UpdateRecord
```

Remarks

This method enables 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:** For backward compatibility with earlier .VBX control, this method is also available as the method of data control's `Recordset`.

See Also:

- Edit Method on page 10-134
- Recordset Property on page 14-29
- Update Method on page 10-257
- Validate Event on page 12-9

ZOrder Method

Applies To

Oracle Data Control on page 1-4

Description

Places a specified form or control at the front or back of the z-order within its graphical level.

14

Data Control Properties

This chapter describes the Oracle Data Control Properties. For an introduction to Data Control, see "Oracle Data Control" on page 1-4.

> **See Also:** For more information, see the Microsoft Visual Basic help and documentation.

This chapter contains these topics:

- Recordset Property
- RecordSource Property
- Session Property
- Tag Property
- Top Property
- TrailingBlanks Property
- Visible Property
- Width Property

The following properties apply to the `OraDynaset` object and to the Oracle Data Control.

- CacheBlocks Property on page 11-16
- CacheSliceSize Property on page 11-20
- CacheSlicesPerBlock Property on page 11-21
- FetchLimit Property on page 11-61
- FetchSize Property on page 11-62

AllowMoveLast Property

Applies To

Oracle Data Control on page 1-4

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 Control's `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.

Datatype

`Integer` (Boolean)

See Also:

- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- Refresh Method on page 10-225

AutoBinding Property

Applies To

Oracle Data Control on page 1-4

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)

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 containing the Oracle Data Control named `oradata1`, 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 creatingthe 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.Refresh

 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
```

See Also:
- Add Method on page 10-8
- AutoBindDisable Method on page 10-39
- AutoBindEnable Method on page 10-41
- CreateDynaset Method on page 10-85
- ExecuteSQL Method on page 10-144
- OraParameter Object on page 9-50
- OraParameters Collection on page 9-68
- RecordSource Property on page 14-31
- Refresh Method on page 13-4

BackColor Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the background color of an object.

See Also: ForeColor Property on page 14-18

Caption Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the text displayed in or next to a control.

Connect Property

Applies To

Oracle Data Control on page 1-4

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 on page 10-212
- OraSession Object on page 9-58
- Refresh Method on page 13-4

Database Property

Applies To

Oracle Data Control on page 1-4

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 on page 9-28
- Refresh Method on page 13-4

DatabaseName Property

Applies To

Oracle Data Control on page 1-4

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:Oracle10:demo"
"x:orasrv"
"mydbalias"(Where mydbalias represents "t:mfg:prod")
```

Data Type

`String`

See Also:

- OpenDatabase Method on page 10-212
- OraSession Object on page 9-58
- Refresh Method on page 13-4

DirtyWrite Property

Applies To

Oracle Data Control on page 1-4

Description

Determines whether or not `Update` and `Delete` `will` or will not check for read inconsistencies.

Usage

```
oradata1.DirtyWrite = [ True | False ]
```

Data Type

`Integer` (Boolean)

Remarks

By default, `DirtyWrite` is `False`, meaning that read consistency will be maintained for `Update` and `Delete` operation on underlying recordset/dynaset object. Changing this property has no effect until a `Refresh` method is sent to the data control.

DragIcon Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the icon to be displayed as the pointer in a drag-and-drop operation.

See Also:

- Drag Method on page 13-2
- DragDrop Event on page 12-2
- DragMode Property on page 14-14
- DragOver Event on page 12-3

DragMode Property

Applies To

Oracle Data Control on page 1-4

Description

Determines manual or automatic dragging mode for a drag-and-drop operation.

See Also:

- Drag Method on page 13-2
- DragDrop Event on page 12-2
- DragIcon Property on page 14-13
- DragOver Event on page 12-3

EditMode Property

Applies To

Oracle Data Control on page 1-4

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 possible `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 `oraconst.txt` file and are intended to match similar constants in the Visual Basic `oraconst.txt` file.

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

Data Type

`Integer`

See Also:

- AddNew Method on page 10-21
- Edit Method on page 10-134
- Update Method on page 10-257

Enabled Property

Applies To

Oracle Data Control on page 1-4

Description

Determines whether the control can respond to user-generated events.

See Also: Visible Property on page 14-37

Font Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the font object to be used for text displayed in a data control.

Usage

```
Oradata1.Font.Bold = True
```

ForeColor Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the foreground color used to display text and graphics in an object.

See Also: BackColor Property on page 14-7

Height Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the height dimension of an object.

See Also:

- Left Property on page 14-21
- Move Method on page 13-3
- Top Property on page 14-35
- Width Property on page 14-38

Index Property

Applies To

Oracle Data Control on page 1-4

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 Also:

Tag Property on page 14-34

Left Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the distance between the internal left edge of an object and the left edge of its container.

See Also:

- Move Method on page 13-3
- Top Property on page 14-35

MousePointer Property

Applies To

Oracle Data Control on page 1-4

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 Also:

- DragIcon Property on page 14-13
- MouseMove Event on page 12-6

Name Property

Applies To

Oracle Data Control on page 1-4

Description

Specifies the name used in code to identify a form, control, or data access object. Not available at run time.

NoRefetch Property

Applies To

Oracle Data Control on page 1-4

Description

By default, `NoRefetch` is `False`, this means that default data set by Oracle Database will not be refetched to the local cache. If the `ORADB_NO_REFETCH` option is `True`, by default, the underlying recordset or dynaset will inherit this property.

Changing this property has no effect until a `Refresh` method is sent to the data control.

Usage

```
oradata1.NoRefetch = [ True | False ]
```

Data Type

`Integer` (Boolean)

Options Property

Applies To

Oracle Data Control on page 1-4

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
```

Data Type

`Long Integer`

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 Database 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 the SQL `"SELECT ... FOR UPDATE"` statement 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"` statement, waits until the row is unlocked before proceeding.

  Optionally, you can 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`.

- No Refetch mode

  In this mode `NULL`s are not explicitly inserted as in the `ORADB_ORAMODE`. In `ORADB_NO_REFETCH` mode, performance is boosted, because data is not refetched to the local cache.

Options Property Flag Values

The `Options` property flag values are:

Constant	Value	Description
ORADB_DEFAULT	&H0&	Accepts the default behavior.
ORADB_ORAMODE	&H1&	Lets Oracle Database set default field (column) values.
ORADB_NOWAIT	&H2&	Does not wait on row locks when executing a SQL "SELECT . . . FOR UPDATE" statement.

These values can be found in the oraconst.txt file. Options may be combined by adding their respective values.

This property is the same as the options passed to the OpenDatabase method. Just as with OpenDatabase, these options affect the OraDatabase object and all associated dynasets created from that database.

Changing this property does not take effect until a Refresh method is sent to the data control.

See Also:

- AddNew Method on page 10-21
- Edit Method on page 10-134
- CreateDynaset Method on page 10-85
- OpenDatabase Method on page 10-212
- OraDatabase Object on page 9-28
- OraDynaset Object on page 9-30
- Refresh Method on page 13-4

OracleMode Property

Applies To

Oracle Data Control on page 1-4

Description

Determine whether the 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.

Usage

```
oradata1.OracleMode = [ True | False ]
```

Data Type

`Integer` (Boolean)

Remarks

This property value is set to `True` by default, which means that fields (columns) changes are reflected in the local cache immediately. Changing this property value has no effect until the `Refresh` method is invoked. If the `ORADB_ORAMODE` mode is used for the database option, the underlying recordset/dynaset inherits this mode.

ReadOnly Property

Applies To

Oracle Data Control on page 1-4

Description

Determines whether the dynaset will be used for read-only operations. Read/write at design time and run time.

Usage

```
oradata1.ReadOnly = [ True | False ]
```

Data Type

`Integer` (Boolean)

Remarks

By default, `ReadOnly` is `False` which means that an attempt will be made to create an updatable dynaset by selecting `ROWID`s from the database. If `ReadOnly` is set to `True`, a non-updatable dynaset is created (`ROWID`s 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, `ROWID`s 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.

See Also:

- CreateDynaset Method on page 10-85
- Refresh Method on page 13-4

Recordset Property

Applies To

Oracle Data Control on page 1-4

Description

Returns a dynaset defined by the data control's `Connect`, `DatabaseName`, and `RecordSource` properties. Not available at design time and read and write at run time.

Usage

```
Set oradynaset = oradata1.Recordset
Set oradata1.Recordset = Oradynaset
```

Data Type

OLE Object (`OraDynaset`)

Remarks

The properties and methods of this dynaset are the same as those of any other dynaset object. The `Recordset` property of the Oracle Data Control (`.OCX`) can be set to external dynaset, or the Recordset property of the other data control. After the setting, Oracle Data control Database, session, and options properties now set to the corresponding properties of the external dynaset. Oracle data control shares the advisories of the external dynaset. This is very useful when attaching dynaset returned from the PL/SQL cursor by `CreatePlsqlDynaset` Method.

Example

This example demonstrates setting `Recordset` property to external dynaset created by `CreatePlsqlDynaset` method. This example returns a PL/SQL cursor as a external dynaset for the different values of `DEPTNO` parameter. Make sure that corresponding stored procedure (found in `EMPCUR.SQL`) is available in the Oracle Database. Copy this code into the definition section of a form containing the Oracle Data Control named `oradata1`. Then, press **F5**.

```
Sub Form_Load ()

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

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

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

' Create the Deptno parameter
 OraDatabase.Parameters.Add "DEPTNO", 10, ORAPARM_INPUT
```

```
 OraDatabase.Parameters("DEPTNO").ServerType = ORATYPE_NUMBER

' Create OraDynaset based on "EmpCursor" created in stored procedure.
 Set OraDynaset = OraDatabase.CreatePLSQLDynaset("Begin Employee.GetEmpData
(:DEPTNO,:EmpCursor); end;", "EmpCursor", 0&)

' Now attach the Oradynaset to Data control's recordset.
  set oradata1.recordset = OraDynaset

...
'  Do some operation
...

 ' Now set the deptno value to 20
 OraDatabase.Parameters("DEPTNO").Value = 20

 'Refresh the sqlstmt
 Oradata1.recordset.Refresh

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

 End Sub
```

See Also:

- Connect Property on page 14-9
- DatabaseName Property on page 14-11
- MoveFirst, MoveLast, MoveNext, and MovePrevious Methods on page 10-199
- OraDynaset Object on page 9-30
- OraFields Collection on page 9-67
- OraParameters Collection on page 9-68
- RecordSource Property on page 14-31

RecordSource Property

Applies To

Oracle Data Control on page 1-4

Description

The SQL `SELECT` statement to be used to create the data control's `RecordSet`. Read/write at design time and run time.

Usage

```
oradata1.RecordSource = [ SQL SELECT Statement ]
```

Data Type

`String`

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.

Whether or not the resultant dynaset can be updated depends on the Oracle SQL rules of updatability, the access you have been granted, and 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.
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 dynaset's `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 to add parameters. The `RecordSource` property can then be set at run time, making use of the automatic binding of database parameters.

Changing this property and calling the `Refresh` method of the `RecordSet` property will create a new dynaset object, but the old dynaset continues to be available for use until all references to it are removed.

Example

This example demonstrates the use of SQL bind variables (parameters) in the `RecordSource` property of the data control. To run this demonstration, 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.Refresh

End Sub
```

See Also:

- Connect Property on page 14-9
- DatabaseName Property on page 14-11
- OraConnection Object on page 9-27
- OraDatabase Object on page 9-28
- OraDynaset Object on page 9-30
- OraParameters Collection on page 9-68
- OraSession Object on page 9-58
- Recordset Property on page 14-29
- Refresh Method on page 13-4
- Updatable Property on page 11-171

Session Property

Applies To

Oracle Data Control on page 1-4

Description

The session object associated with the data control. Not available at design time and read-only at run time.

Usage

```
orasession = oradata1.Session
```

Data Type

OLE Object (`OraSession`)

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.

See Also:

- OraDatabase Object on page 9-28
- OraSession Object on page 9-58
- OraSessions Collection on page 9-69

Tag Property

Applies To

Oracle Data Control on page 1-4

Description

Stores any extra data needed by your application.

Top Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the distance between the internal top edge of an object and the top edge of its container.

See Also:

- Move Method on page 13-3
- Left Property on page 14-21

TrailingBlanks Property

Applies To

Oracle Data Control on page 1-4

Description

Determines whether trailing blanks should be removed from character string data retrieved from the database. Read/write at design time and run time.

Usage

```
oradata1.TrailingBlanks = [ True | False ]
```

Data Type

`Integer` (Boolean)

Remarks

By default, `TrailingBlanks` is `False`. This means that trailing blanks will be removed from character string data retrieved from the database.

Changing this property has no effect until a `Refresh` method is sent to the data control.

See Also:

- CreateDynaset Method on page 10-85
- Refresh Method on page 13-4

Visible Property

Applies To

Oracle Data Control on page 1-4

Description

Determines whether an object is visible or hidden.

See Also: Enabled Property on page 14-16

Width Property

Applies To

Oracle Data Control on page 1-4

Description

Determines the width dimension of an object.

See Also:

- Height Property on page 14-19
- Left Property on page 14-21
- Move Method on page 13-3
- Top Property on page 14-35

A

Appendix A

This appendix includes the following:

Oracle Data Types

The following are code for Oracle data types.

Table A–1 Oracle Data Type Codes

Oracle Data Type	Codes
ORATYPE_VARCHAR2	1
ORATYPE_NUMBER	2
ORATYPE_SINT	3
ORATYPE_FLOAT	4
ORATYPE_STRING	5
ORATYPE_DECIMAL	7
ORATYPE_LONG	8
ORATYPE_VARCHAR	9
ORATYPE_DATE	12
ORATYPE_REAL	21
ORATYPE_DOUBLE	22
ORATYPE_UNSIGNED8	23
ORATYPE_RAW	23
ORATYPE_LONGRAW	24
ORATYPE_UNSIGNED16	25
ORATYPE_UNSIGNED32	26
ORATYPE_SIGNED8	27
ORATYPE_SIGNED16	28
ORATYPE_SIGNED32	29
ORATYPE_PTR	32
ORATYPE_OPAQUE	58

Table A–1 (Cont.) Oracle Data Type Codes

Oracle Data Type	Codes
ORATYPE_UINT	68
ORATYPE_CHAR	96
ORATYPE_CHARZ	97
ORATYPE_BFLOAT	100
ORATYPE_BDOUBLE	101
ORATYPE_CURSOR	102
ORATYPE_ROWID	104
ORATYPE_MLSLABEL	105
ORATYPE_OBJECT	108
ORATYPE_REF	110
ORATYPE_CLOB	112
ORATYPE_BLOB	113
ORATYPE_BFILE	114
ORATYPE_CFILE	115
ORATYPE_RSLT	116
ORATYPE_NAMEDCOLLECTION	122
ORATYPE_COLL	122
ORATYPE_TIMESTAMP	187
ORATYPE_TIMESTAMPTZ	188
ORATYPE_INTERVALYM	189
ORATYPE_INTERVALDS	190
ORATYPE_SYSFIRST	228
ORATYPE_TIMESTAMPLTZ	232
ORATYPE_SYSLAST	235
ORATYPE_OCTET	245
ORATYPE_SMALLINT	246
ORATYPE_VARRAY	247
ORATYPE_TABLE	248
ORATYPE_OTMLAST	320
ORATYPE_RAW_BIN	2000

These codes are also listed in the `oraconst.txt` file located in the *ORACLE_BASE*`\`*ORACLE_HOME*`\oo4o` directory.

Additional Schemas

Occasionally other schemas are required to run examples. These schemas are listed in the following sections.

Schema Objects Used in OraMetaData Examples

This section presents OraMetaData schema definitions.

```
CREATE TYPE ORAMD_ADDRESS AS OBJECT ( no NUMBER,
                  street VARCHAR(60),
                  state CHAR(2),
                  zip CHAR(10),
MEMBER PROCEDURE ChangeStreetName(newstreet IN VARCHAR2)
);
```

Schema Objects Used in LOB Data Type Examples

The following schema objects are used in the OraLOB and `BFILE` examples. Run the SQL script `ORAEXAMP.SQL` on your database to set up the schema.

```
CREATE TABLE part (
                  part_id NUMBER,
                  part_name VARCHAR2(20),
                  part_image BLOB,
                  part_desc CLOB,
                  part_collateral BFILE
);

Create Directory NewDirectoryName as 'C:\valid\path'
```

Schema Objects Used in the OraObject and OraRef Examples

The following schema objects are used in the `OraObject` and `OraRef` examples. Data for the following tables can be inserted with the `ORAEXAMP.SQL` script that is provided with the OO4O installation.

```
CREATE TYPE address AS OBJECT (
                        street VARCHAR2(200),
                        city VARCHAR2(200),
                        state CHAR(2),
                        zip VARCHAR2(20)
);

CREATE TYPE person as OBJECT(
                        name VARCHAR2(20),
                        age NUMBER,
                        addr ADDRESS);

CREATE TABLE person_tab of PERSON;
CREATE TABLE customers(
                        account NUMBER,
                        aperson REF PERSON);
```

Schema Objects Used in OraCollection Examples

The following schema is used in examples of `OraCollection` methods

```
CREATE TYPE ENAMELIST AS VARRAY(20) OF VARCHAR2(30);
  CREATE TABLE department (
    dept_id NUMBER(2),
    name VARCHAR2(15),
    ENAMES ENAMELIST);
```

```
  DROP TYPE COURSE;

CREATE TYPE Course AS OBJECT (
  course_no NUMBER(4),
  title VARCHAR2(35),
  credits NUMBER(1));

CREATE TYPE CourseList AS TABLE OF Course;

CREATE TABLE division (
  name VARCHAR2(20),
  director VARCHAR2(20),
  office VARCHAR2(20),
  courses CourseList)
NESTED TABLE courses STORE AS courses_tab;
```

Glossary

BFILEs

External binary files that exist outside the database tablespaces residing in the operating system. BFILEs are referenced from the database semantics, and are also known as external LOBs.

Binary Large Object (BLOB)

A large object data type whose content consists of binary data. Additionally, this data is considered raw as its structure is not recognized by the database.

Character Large Object (CLOB)

The LOB data type whose value is composed of character data corresponding to the database character set. A `CLOB` may be indexed and searched by the Oracle Text search engine.

coordinated universal time (UTC)

UTC was formerly known as Greenwich Mean Time.

Large Object (LOB)

The class of SQL data type that is further divided into internal LOBs and external LOBs. Internal LOBs include `BLOB`s, `CLOB`s, and `NCLOB`s while external LOBs include `BFILE`s.

National Character Large Object (NCLOB)

The LOB data type whose value is composed of character data corresponding to the database national character set.

PL/SQL

Oracle procedural language extension to SQL.

primary key

The column or set of columns included in the definition of a table's PRIMARY KEY constraint.

UTC

UTC (Coordinated Universal Time) was formerly known as Greenwich Mean Time.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z