Table of Contents

1 What's New

New Features in ODAC 13.2

- Added support for RAD Studio 12 Athens Release 1
- Added support for Lazarus 3.2
- Added the GetAuditActionBanner function to TOraSession
- Added the GetUnauthorizedAccessBanner function to TOraSession

New Features in ODAC 13.1

- Added support for Lazarus 3.0
- Added support for connect to servers with enabled Security Banners in the Direct mode
- Improved error message about a non-existent database object in another schema

New Features in ODAC 13.0

- Added support for RAD Studio 12
- Added support for Oracle 23c
- Added support for macOS Sonoma
- Added support for iOS 17
- Added support for Android 13
- Added support for nested Macros in SQL queries
- Added support Display Format for Aggregate fields
- Added SHA-2(SHA-256, SHA-512) in hash algorithm for encryption
- Improved UnicodeEnvironment support for non-Unicode Delphi versions

New Features in ODAC 12.3

- Added support for RAD Studio 11 Alexandria Release 3
- Added support for iOS Simulator ARM 64-bit target platform
- Added support for Lazarus 2.2.6
- Added support for the YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, GETDATE, DATE,

TIME, TRIM, TRIMLEFT, TRIMRIGHT statements in TDADataSet.Filter

- Added support for the mathematical operations in TDADataSet.Filter
- Added support for Aggregate Fields and InternalCalc Fields
- Added ability to restore from file with TEncoding via the Dump component
- Improved detection of home directories in recent versions of Oracle
- Now the SetRange will function according to the case sensitivity of keywords in IndexFieldNames
- Now valid exception will be raised instead of AV when memory can't be allocated for the large row count

New Features in ODAC 12.2

- Added support for RAD Studio 11 Alexandria Release 2
- Added support for Lazarus 2.2.2
- Added support for iOS 15
- Added support for Android 12
- Added the CloneCursor method for Query and Table components that allows sharing data between datasets
- Improved the performance of exporting to XML
- Fixed bug when a connection string parameter value contains a single quote

New Features in ODAC 12.1

- RAD Studio 11 Alexandria Release 1 is supported
- Lazarus 2.2.0 is supported
- Windows 11 is supported
- macOS Monterey is supported

New Features in ODAC 12.0

- RAD Studio 11 Alexandria is supported
- macOS ARM is supported

- Added demo project for FastReport FMX

New Features in ODAC 11.4

- Oracle 21c is supported
- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- Performance of batch operations is improved
- Performance of the FindFirst, FindNext, FindLast, and FindPrior methods is improved
- The PrefetchRows option in the Direct mode is supported
- Data fetch performance in the Direct mode is improved
- LOB read/write performance is improved

New Features in ODAC 11.3

- Oracle 20c is supported
- Connection via SSL protocol is supported
- Connection via SSH protocol is supported
- Connection via HTTP tunnel is supported
- Lazarus 2.0.10 and FPC 3.2.0 are supported
- Performance of Batch Insert, Update, and Delete operations is improved

New Features in ODAC 11.2

- RAD Studio 10.4 Sydney is supported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported
- Mapping the FLOAT Oracle data type to the ftNumber field is added

New Features in ODAC 11.1

- Android 64-bit is supported
- Lazarus 2.0.6 is supported
- Oracle 19c is supported
- Now Trial edition for macOS and Linux is fully functional
- Long database object names is supported

New Features in ODAC 11.0

- macOS 64-bit is supported
- Release 2 for RAD Studio 10.3 Rio, Delphi 10.3 Rio, and C++Builder 10.3 Rio is now required

New Features in ODAC 10.4

- Lazarus 2.0.2 is supported
- CommandTimeout is supported
- Support for ChangePassword in the Direct mode is improved
- The DefaultSortType property for TVirtualTable is added
- Performance of the SaveToFile/LoadFromFile methods of TVirtualTable is significantly increased

New Features in ODAC 10.3

- RAD Studio 10.3 Rio is supported
- Oracle 18c is supported
- Implicit result sets in Oracle 12 are supported
- The OpenNext method for opening a next cursor or implicit result set is added
- Support of UPPER and LOWER functions for Unified SQL is added

New Features in ODAC 10.2

- Lazarus 1.8.4 is supported
- Performance of data fetching in the Direct mode is improved

- Performance of describing stored procedures in the Direct mode is improved
- Performance of batch operations is improved
- Demo projects for IntraWeb 14 are added
- Now the TOraTimeStamp.AsDateTime method returns TDateTime value with milliseconds
- Now non-compiled stored procedures can be described in the Direct mode
- Performance of describing stored procedures in the Direct mode is improved
- Support for TIMESTAMP WITH TIMEZONE in the Direct mode is improved

New Features in ODAC 10.1

- Oracle 12c connection modes (SYSBACKUP, SYSDG, SYSKM) in the Direct mode are supported
- OS authentication in the Direct mode is supported
- NChar literal replacement is supported

New Features in ODAC 10.0

- RAD Studio 10.2 Tokyo is supported
- Linux in RAD Studio 10.2 Tokyo is supported
- Lazarus 1.6.4 and Free Pascal 3.0.2 is supported
- Oracle Encryption in the Direct mode is supported
- Oracle Data Integrity in the Direct mode is supported
- Oracle Cloud (DBaaS) in the Direct mode is supported
- Oracle 12c authentication in the Direct mode is supported
- SECUREFILE in the Direct mode is supported
- Prefetch LOBs for Oracle 11 and higher is supported
- EDITIONABLE and NONEDITIONABLE clause is supported
- The PrefetchLobSize option is added
- ANYDATA is supported
- Field size detecting for servers with multi-byte charset when UseUnicode=False is

improved

- Now NUMBER data type without fixed scale has precision=39 and scale=39 instead of 38

New Features in ODAC 9.7

- RAD Studio 10.1 Berlin is supported
- Lazarus 1.6 and FPC 3.0.0 is supported
- Support for the BETWEEN statement in TDADataSet.Filter is added
- Data Type Mapping performance is improved
- Performance of TDALoader on loading data from TDataSet is improved

New Features in ODAC 9.6

- RAD Studio 10 Seattle is supported
- Now NULL and empty strings are different values for ftOraLob and ftOraClob parameters
- Now Trial for Win64 is a fully functional Professional Edition
- Support for Offset is added for DML arrays
- Support for OraNet.PacketSize is added to improve performance in VPN and Wireless networks
- Support for Object References in the Direct mode is added
- Support for Object attributes with the XML data type is added

New Features in ODAC 9.5

- RAD Studio XE8 is supported
- AppMethod is supported
- Direct mode in Lazarus is supported
- Now the Direct mode is supplied as source code
- Support for Objects in the Direct mode is added
- Support for XML in the Direct mode is added
- Support for EZCONNECT in the Direct mode is added
- Support for fields with Cursor data type in the Direct mode is added

- Now statements with RETURN INTO clauses can return RowsAffected in the Direct mode

New Features in ODAC 9.4

- RAD Studio XE7 is supported
- Lazarus 1.2.4 is supported
- RAC server support is improved
- Support for WITH FUNCTION clause for Oracle 12c is added
- GetServerList doesn't cut the WORLD postfix anymore
- The HideRowId option is added
- Demo projects for FastReport 5 are added
- The TCustomDADataSet.GetKeyFieldNames method is added
- The ConstraintColumns metadata kind for the TDAMetadata component is added
- Workaround for the bug with calling Halt in the OnCreate event is added

New Features in ODAC 9.3

- RAD Studio XE6 is supported
- Android in C++Builder XE6 is supported
- Lazarus 1.2.2 and FPC 2.6.4 is supported
- SmartFetch mode for TDataSet descendants is added
- Now update queries inside TDataSet descendants have correct owner
- Possibility to assign external SvcCtx to connection is added
- DataTypeMapping conversion from XMLType to ftString is added
- DataTypeMapping conversion from Interval to ftString is added
- The TOraDataSetOptions.MasterFieldsNullable property is added

New Features in ODAC 9.2

- iOS in C++Builder XE5 is supported
- RAD Studio XE5 Update 2 is now required
- Now .obj and .o files are supplied for C++Builder

- An ability to establish OCI and Direct connections in the same application is supported
- New Oracle 12c connection modes are added (SYSBACKUP, SYSDG, SYSKM)
- The AsTimeStamp property is added to the TOraTimeStamp class
- Compatibility of migrating floating-point fields from other components is improved

New Features in ODAC 9.1

- RAD Studio XE5 is supported
- Application development for Android is supported
- Lazarus 1.0.12 is supported
- Performance is improved
- Automatic checking for new versions is added
- Flexible management of conditions in the WHERE clause is added
- The possibility to use conditions is added
- IPv6 protocol support is added
- The possibility to use ranges is added
- Support of the IN keyword in the TDataSet.Filter property is added
- Like operator behaviour when used in the Filter property is now similar to TClientDataSet
- The AllowImplicitConnect option for the TOraSession component is added
- The SQLRecCount property for the TOraQuery and TOraStoredProc components is added
- The ScanParams property for the TOraScript component is added
- The RowsAffected property for the TOraScript component is added
- The UROWID data type is supported in the Direct mode

New Features in ODAC 9.0

- Rad Studio XE4 is supported
- NEXTGEN compiler is supported
- Application development for iOS is supported
- FPC 2.6.2 and Lazarus 1.0.8 are supported

- BINARY_DOUBLE & BINARY_FLOAT data types support in the Direct mode is added
- Connection string support is improved
- Possibility to encrypt entire tables and datasets is added
- Possibility to determine if data in a field is encrypted is added
- Support for TimeStamp, Single and Extended fields in VirtualTable is added

New Features in ODAC 8.6

- Rad Studio XE3 Update 1 is now required
- C++Builder 64-bit for Windows is supported

New Features in ODAC 8.5

- Rad Studio XE3 is supported
- Windows 8 is supported

New Features in ODAC 8.2

- Update 4 Hotfix 1 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Data Type Mapping support is added
- Data Encryption in a client application is added
- The TOraEncryptor component for data encryption is added
- Integration with dbForge Studio for Oracle is added
- Calling of the TCustomDASQL.BeforeExecute event is added
- FieldType ftOraTimeStamp is added

New Features in ODAC 8.1

- Update 2 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Mac OS X and iOS in RAD Studio XE2 is supported
- FireMonkey support is improved
- Lazarus 0.9.30.2 and FPC 2.4.4 are supported
- Mac OS X in Lazarus is supported

- Linux x64 in Lazarus is supported
- FreeBSD in Lazarus is supported
- Oracle 11 Express Edition is supported
- Support for the NonBlocking option is added
- The QueryResultOnly option is added to TOraChangeNotification

New Features in Oracle Data Access Components 8.00

- Embarcadero RAD Studio XE2 is supported
- Application development for 64-bit Windows is supported
- FireMonkey application development platform is supported
- Support of master/detail relationship for TVirtualTable is added
- OnProgress event in TVirtualTable is added
- TDADataSetOptions.SetEmptyStrToNull property that allows inserting NULL value instead of empty string is added

New Features in Oracle Data Access Components 7.20

- Lazarus 0.9.30 and FPC 2.4.2 is supported
- Oracle 9, Oracle 10, and Oracle 11 authentication in the Direct mode is supported
- Case sensitive login and password in the Direct mode is supported
- Unicode login and password in the Direct mode is supported
- Client Identifier in the Direct mode is supported
- Support of BLOB, CLOB, and NCLOB data types in ToraLoader is improved
- Support of "table of blob/clob" data type is improved

New Features in Oracle Data Access Components 7.10

- Support for connection with using Service Name in the Direct mode
- Support for the ChangePassword functionality in the Direct mode
- Improved returning cursors with different fields list from TOraStoredProc
- Search for the TNS_ADMIN variable in the Oracle root location in the registry

- Checking that dataset is open on calling the TDataSet.Locate method

New Features in Oracle Data Access Components 7.00

- Embarcadero RAD Studio XE suppored
- Support of National parameter to Package Wizard
- Ability to lock records in the CachedUpdate mode
- Ability to send call stack information to the dbMonitor component
- Added ability to lock records in the CachedUpdate mode
- Added OnStart, OnCommit, OnRollback events to TDATransaction
- Added OnInfoMessage event
- Added support for dbMonitor 3
- Added support for using user-defined UpdatingTable when ODAC cannot detect a table list for a query
- Updated Oracle client version detection for Linux by OCI API
- Changed the LocateEx method behavior: now LocateEx centers records equal to Locate
- Now Required flag is set for UpdatingTable fields only
- Now the AssignConnect method copies transaction state

New Features in Oracle Data Access Components 6.90

- Embarcadero RAD Studio 2010 supported
- Support for BINARY_FLOAT and BINARY_DOUBLE data types in TOraArray
- Support for reading a comma separated list of aliases from TNSNAMES.ORA
- Support for ALTER .. COMPILE in GetCompilationError
- Unicode support in CLOB attributes of OBJECT type
- Added EnableOraTimeStamp option of TOraSession
- Added distinction between empty string and null value when saving/loading string fields in TVirtualTable
- Added support for Free Pascal under Linux

- Added NoPreconnect property to TOraScript for executing CONNECT and CREATE DATABASE commands
- The Disconnected property to TCustomDADataSet
- Now the value from the master dataset has priority over the DefaultExpression value

New Features in Oracle Data Access Components 6.80

- Free Pascal under Linux supported
- Added NoPreconnect property to TOraScript for executing CONNECT and CREATE DATABASE commands

New Features in Oracle Data Access Components 6.70

- Delphi 2009 and C++Builder 2009 supported
- Extended Unicode support for Delphi 2007 added (special Unicode build)
- Free Pascal 2.2 supported
- Powerful design-time editors implemented in Lazarus
- Optimized LOB processing in Direct mode
- Completed with more comprehensive structured Help

New Features in Oracle Data Access Components 6.50

- Improved support of default field values
- The new component for metadata receiving added
- Added support of TWideMemoField
- The BCD types supported

New Features in Oracle Data Access Components 6.25

- Oracle 11g supported

New Features in Oracle Data Access Components 6.20

- CodeGear RAD Studio 2007 supported
- Added ability to customize whether update SQL statements should be prepared

- Added ability to set number of rows to be prefetched by OCI
- Added the OnProgress event in TOraLoader

New Features in Oracle Data Access Components 6.10

- C++Builder 2007 supported

New Features in Oracle Data Access Components 6.05

- Added Oracle Package Wizard that simplifies working with PL/SQL Packages

New Features in Oracle Data Access Components 6.00

New functionality:

- Delphi 2007 for Win32 supported
- Implemented Disconnected Model for working offline and automatically connecting and disconnecting
- Implemented Local Failover for detecting connection loss and implicitly re-executing some operations
- LargeInt fields supported
- WideMemo field type in Delphi 2006 supported
- Added DataSet Manager to control project datasets
- Integration with OraDeveloper Tools 2.00 added
- New TCRBatchMove component for transferring data between all types of TDataSet descendants added
- Data export and M:Devart.Dac.TVirtualTable.LoadFromFile(System.String,System.Boolean) to/from XML supported
- Support for sending messages to DBMonitor from any point in your program added

Support for more Oracle server functionality:

- Distributed transactions supported
- Advanced Queuing support added

- Change notifications functionality of Oracle 10g supported
- DBMS_TRACE package and SQLTrace functionality supported
- OCI Connection Pooling, Statement Caching, and Proxy Session Pooling added
- External Procedures support added

Extensions and improvements to existing functionality:

- General performance improved
- Master/detail functionality extensions:
  - Local master/detail relationship support added
  - Support for master/detail relationships in CachedUpdates mode added
- Connection pool functionality improvements:
  - Efficiency significantly improved
  - API for draining the connection pool added
- TOraScript component improvements:
  - Support for executing individual statements in scripts added
  - Support for executing huge scripts stored in files with dynamic loading added
  - Ability to use standard SQL*Plus tool syntax added
- Greatly increased performance of applying updates in CachedUpdates mode
- Working with calculated and lookup fields improvements:
  - Local sorting and filtering added
  - Record location speed increased
  - Improved working with lookup fields
- Ability to customize update commands by attaching external components to TOraUpdateSQL objects added
- Support for using BeforeFetch and AfterFetch events in NonBlocking mode added
- Temporary LOBs for updating LOB fields supported

- Support for setting connection timeout values for Direct mode added
- Ability to include all fields in automatically generated update SQLs added
- Support for default field value expressions added

Usability improvements:

- Syntax highlighting in design-time editors added
- Completely restructured and clearer demo projects

New Features in Oracle Data Access Components 5.70

- Delphi 2006 supported

New Features in Oracle Data Access Components 5.55

- Added ability to automatically prepare queries (TCustomDADataSet.Options.AutoPrepare)
- Added ability to synchronize positions in different DataSets (TCustomDADataSet.GotoCurrent)

New Features in Oracle Data Access Components 5.50

- Delphi 2005 supported
- Performance of Net-option improved
- Added Schema property for TOraSession component
- Added ProxySession property for TOraSession component

New Features in Oracle Data Access Components 5.10

- Local sort ability using IndexFieldNames property added
- T:Devart.Odac.OraDataAdapter component for Delphi 8 added

New Features in Oracle Data Access Components 5.00

- Support for Delphi 8 added
- Oracle 10g support added
- Connection pooling supported

- Character conversion supported in Oracle 9i in Direct mode
- Unicode character data supported in Direct mode
- Support TIMESTAMP, INTERVAL data types in Direct mode
- Support for Oracle internal NUMBER datatype added in Direct mode
- Performance improved
- TCRGrid sources in Standard and Net editions
- .NET Windows Forms demo project added
- ASP.NET demo project added

New Features in Oracle Data Access Components 4.50

- XMLTYPE datatype support added
- WideString support added to work with Unicode character data
- Transparent Application Failover support added

New Features in Oracle Data Access Components 4.15

- Support for Oracle internal NUMBER datatype added. Allows to work with high precision numbers without accuracy losses

New Features in Oracle Data Access Components 4.10

- Support for Oracle 9i NOT FINAL objects added
- TIMESTAMP and INTERVAL support for Oracle objects added

25-Dec-02 Oracle Data Access Components 4.05 new features:

- Transaction control schema changed. Now TOraSession.InTransaction shows actual user transaction state on server (implicit commit and rollback are considered).
- DBMonitor client implementation moved to COM server. Now ODAC is incompatible with DBMonitor 2.02 or lower.
- LOB attributes support for object fields added
- Temporary LOBs support added

- Constants ftTimeStampTZ and ftTimeStampLTZ added. Used in TOraTimeStampField.
- UROWID support for index organized tables added
- Option ConvertEOL added

New Features in Oracle Data Access Components 4.00

- Delphi 7 support
- Kylix 3 for C++ support
- Oracle 9 scrollable cursors support
- New memory management model for ftString and ftVarBytes types. Allows significantly decrease memory usage on large tables fetch. Controlled by FlatBuffers dataset option;
- RAW datatype support (as ftVarBytes)
- Support for complex fields (blobs, objects etc.) in CachedUpdates mode
- New 'Prepare' schema. Now if user does not explicitly call Prepare method before opening dataset there is no additional roundtrip to server for select-list describe (OCIStmtExecute(DESCRIBE_ONLY) call). I.e. Open (Execute for SELECT) without Prepare is performed in a more optimal way.

New Features in Oracle Data Access Components 3.90

- Kylix 3 support

New Features in Oracle Data Access Components 3.85

- DBMonitor support
- New version of OraTools support (v. 2.50)

New Features in Oracle Data Access Components 3.80

- Oracle9 timestamp and interval datatypes support
- Performance optimization for queries with many fields, especially for TSmartQuery and TOraTable
- Runtime packages division for Delphi6, C++Builder6, Kylix, Kylix2, see manual
- Auto generation RETURNING clause for LOBs added to design-time component editor

New Features in Oracle Data Access Components 3.60

- supports C++Builder 6

New Features in Oracle Data Access Components 3.50

- supports Kylix 2
- multibytes charsets support
- direct lob access support
- using OraTools Add-in

New Features in Oracle Data Access Components 3.30

- supports Oracle 9i
- Net edition for Kylix

New Features in Oracle Data Access Components 3.20

- supports Delphi 6
- new version of OraDesigner
- OraExplorer
- printed documentation
- BDE Migration Wizard

New Features in Oracle Data Access Components 3.00

- using standard TParam object
- separate run- and design-time packages
- get original name of fields
- retrieve field's default value
- in Direct mode support
- Kylix ready

New Features in Oracle Data Access Components 2.50

- supports multiple Oracle Homes
- supports Borland SQL Monitor
- TOraSQLMonitor component
- default session
- customizable connect dialog
- ConnectDialog and Threads demos added

New Features in Oracle Data Access Components 2.20

- supports C++Builder 5
- macros in update SQL

New Features in Oracle Data Access Components 2.10

- customized TSmartQuery data updating
- supports DML array operations
- macros in TOraSQL and TOraScript
- TOraLoader component
- supports Oracle 8 Lite
- easy installation

New Features in Oracle Data Access Components 2.00

- supports Oracle8 Objects
- supports Oracle8 REFs
- supports Oracle8 Arrays
- supports Oracle8 Nested tables
- supports Oracle8 BFiles
- using RETURNING with Oracle8
- smart refreshing
- TOraNestedTable component
- TOraScript component

- TOraAlerter component
- TBFileField component
- TOraFile class
- TOraLob class
- TOraType class
- TOraObject class
- TOraRef class
- TOraArray class
- TOraNestTable class
- Alerter, Arrays, BFile, BLOBPics, DLL, DMLArray, FetchCursors, NestedTables, Objects, Refs demos added

New Features in Oracle Data Access Components 1.85

- TOraProvider component
- TBDESession component
- Supports Oracle 8i
- C++Builder 4 package

New Features in Oracle Data Access Components 1.75

- TOraTable component
- TStoredProc component

New Features in Oracle Data Access Components 1.70

- Supports BLOB and CLOB data types Oracle 8
- Supports nested tables
- TVirtualTable component
- Embeded SQL Designer with PL/SQL debugger
- C++Builder version

New Features in Oracle Data Access Components 1.50

- Supports native interface Oracle 8.0
- Supports PL/SQL tables
- TOraErrorHandler

 Request Support DAC Forum Provide Feedback

2 General Information

This section contains general information about Oracle Data Access Components

- Overview
- Features
- Requirements
- Compatibility
- Using Several DAC Products in One IDE
- Component List
- Hierarchy Chart
- Editions
- Getting Support
- Frequently Asked Questions

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

2.1 Overview

Oracle Data Access Components (ODAC) is a library of components that provides native connectivity to Oracle from Delphi, C++Builder, Lazarus (and Free Pascal) for 32-bit and 64-bit Windows, macOS, and Linux platforms. The ODAC library is designed to help programmers develop faster, cleaner and more native Oracle database applications. ODAC, a high-performance and feature-rich Oracle connectivity solution, is an efficient native

alternative to the Borland Database Engine (BDE) and standard dbExpress driver. It provides both possibility of connection to Oracle by means of native Oracle data access and direct access to Oracle without Oracle Client.

The ODAC library is actively developed and supported by the Devart Team. If you have questions about ODAC, email the developers at odac@devart.com or visit ODAC online at https://www.devart.com/odac/

Table of Contents

Advantages of ODAC Technology

ODAC is a direct database connectivity wrapper built specifically for the Oracle server. ODAC offers wide coverage of the Oracle feature set, supports both Client and Direct connection modes, and emphasizes optimized data access strategies.

Wide Coverage of Oracle Features

By providing access to the most advanced database functionality, ODAC allows developers to harness the full capabilities of the Oracle and optimize their database applications. ODAC stands out as the set of components with the widest support of Oracle functionality. It is the only component to support Oracle distributed transactions and implements support for controlling statement caching , OCI pooling , and Oracle Advanced Queuing. View the full list of supported Oracle features in Features.

Native Connection Options

ODAC offers two connection modes to the Oracle server: connection through the Oracle Call Interface and direct connection over TCP/IP. ODAC-based database applications are easy to deploy, do not require installation of other data provider layers (such as BDE), and tend to be faster than those that use standard data connectivity solutions. See the How does ODAC Work section.

Oracle Advanced Features Support

ODAC has extra components designed to simplify some tasks and support Oracle-specific technologies. Particularly, OraScript serves to execute series of SQL statements, OraLoader serves to load external data into Oracle databases, OraeAlerter and OraPipe transfer messages and data between connections or client applications, and so on. ODAC includes set of classes designed to work with Oracle Advanced Queuing technology.

Cross-Platform Solution for Delphi, C++Builder, and Lazarus

ODAC is a cross-platform solution for developing applications using various IDEs: RAD Studio, Delphi, C++Builder, Lazarus (and FPC) on Windows, macOS, and Linux, for both x86 and x64 platforms. ODAC also provides support for the FireMonkey platform, which allows you to develop visually spectacular high-performance native applications for Windows and macOS.

Optimized Code

The goal of ODAC is to enable developers to write efficient and flexible database applications. The ODAC library is implemented using optimized code and advanced data access algorithms. Component interfaces undergo comprehensive performance tests and are designed to help you write thin and efficient product data access layers. Find out more about using ODAC to optimize your database applications in Increasing Performance. To see the results of ODAC performance tests, consult the performance comparison section on the ODAC website.

Compatibility with Other Connectivity Methods

The ODAC interface retains compatibility with standard VCL data access components like BDE. Existing BDE-based applications can be easily migrated to ODAC and enhanced to take advantage of Oracle-specific features like alerts, pipes, and the direct-path interface.

Project migration can be automated with the BDE Migration Wizard. Find out more about Migration Wizard in Using Migration Wizard.

Development and Support

ODAC is an Oracle connectivity solution that has been actively developed and supported since 1998. ODAC comes with full documentation, demo projects, and fast (usually within one business day) technical support by the ODAC development team. Find out more about getting help or submit feedback and suggestions to the ODAC development team in Getting Support.

The description of the ODAC components is provided in the Component List.

Key Features

- Direct access to server data. Does not require installation of other data provider layers (such as BDE and ODBC)
- In Direct mode does not require Oracle client software and works directly through TCP/IP
- VCL, LCL and FMX versions of library available
- Support of Windows, macOS, and Linux, for both x86 and x64 platforms.
- Full support of the latest versions of Oracle
- Support for all Oracle data types
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly re-executing certain operations
- Oracle Transparent Application Failover support
- All types of local sorting and filtering , including by calculated and lookup fields
- Automatic data updating with TOraQuery , TSmartQuery , and TOraTable components
- Unicode and national charsets support
- Distributed transaction support
- Oracle Advanced Queuing support
- Support for many Oracle-specific features, such as alerts, pipes and direct path interface
- Advanced script execution functionality with TOraScript component

- Support for using Macros in SQL
- Easy migration from BDE with Migration Wizard
- Lets you use Professional Edition of Delphi and C++Builder to develop client/server applications
- Included annual ODAC Subscription with Priority Support
- Full Oracle Objects support
- Comprehensive REF CURSOR support
- PL/SQL tables and PL/SQL records support
- Operates in both connected and disconnected models
- Typed OraPackage component for wrapping PL/SQL packages
- Auxiliary components for SQL scripts and bulk data transfer
- Ability of monitoring query execution TOraSQLMonitor
- Licensed royalty-free per developer, per team, or per site

The full list of ODAC features can be found in Features.

How does ODAC work?

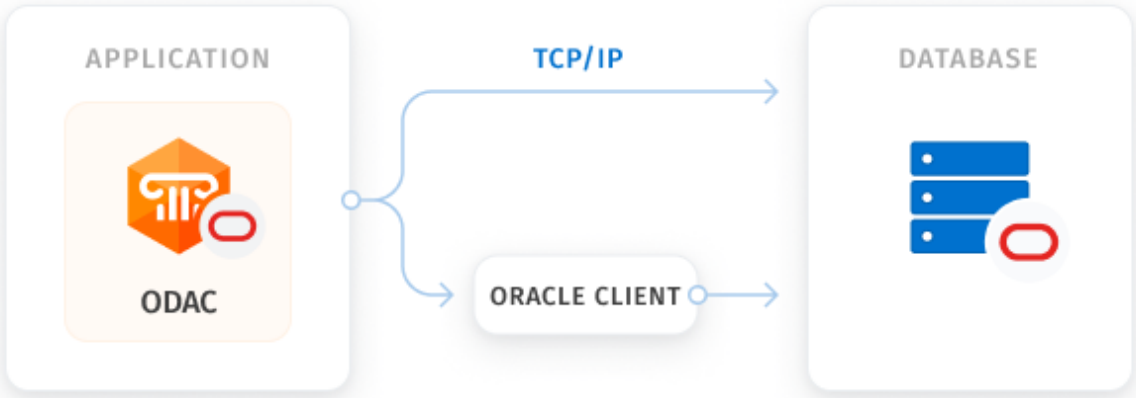
ODAC allows you to connect to Oracle in two ways: in the Direct mode over TCP/IP or in the Client mode using Oracle Client. The Direct mode can be enabled or disabled using the Direct property.

In the Direct mode, ODAC connects to Oracle directly via TCP/IP without using Oracle client software and enables you to build really thin database applications.

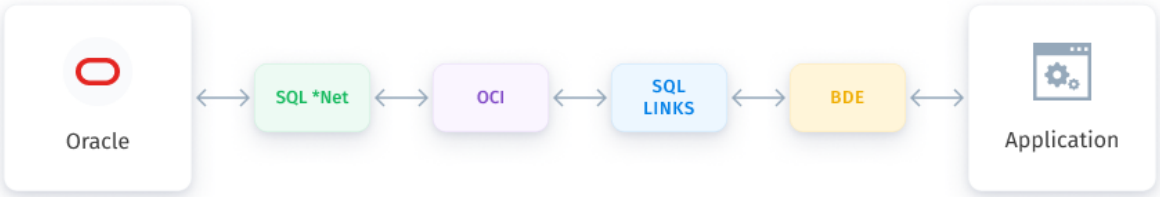
In Client mode, ODAC connects to Oracle through the Oracle Call Interface (OCI). OCI is an application programming interface that allows an application developer to use a third-generation language's native procedure or function calls to access the Oracle database server and control all phases of SQL statement execution. OCI is a library of standard database access and retrieval functions in the form of a dynamic-link library.

In contrast, the Borland Database Engine (BDE) uses several layers to access Oracle and requires additional data access software to be installed on client machines.

ODAC Connection



BDE Connection



Request Support DAC Forum Provide Feedback

2.2 Features

Supported target platforms:

- Windows 32-bit and 64-bit
- macOS 64-bit
- Mac ARM
- iOS 64-bit
- iOS Simulator ARM 64-bit
- Android 32-bit and 64-bit
- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit

General usability:

- Direct access to server data. Does not require installation of other data provider layers (such as BDE and ODBC)
- Interface compatible with standard data access methods, such as BDE and ADO
- VCL, LCL, and FMX versions of library available
- Separated run-time and GUI specific parts allow you to create pure console applications such as CGI
- Unicode and national charset support
- Highly usable design time support
- Easy to deploy

Network and connectivity:

- In Direct mode does not require Oracle client software and works directly through TCP/IP
- Disconnected Model with automatic connection control for working with data offline
- Local Failover for detecting connection loss and implicitly reexecuting certain operations
- Support for setting connection timeout values for Direct mode
- Support for OraNet.PacketSize to improve performance in VPN and Wireless networks
- Secure connections with SSL, SSH, and HTTP/HTTPS tunneling (using SecureBridge)
- Support for OS authentication
- Support for Proxy Authentication
- DBA privileges to open a session with
- Support for Oracle 19c connection modes
- Support for the change expired password
- Connection using Service Name or SID in the Direct mode
- Support for RAC Server
- Support for both IPv6 and Ipv4 protocol
- Support for IFILE in tnsnames.ora

- Support for EZCONNECT connection string

Compatibility:

- Full support of the latest versions of Oracle
- Support for all versions of Oracle Clients, including Instant Client
- Support for all Oracle data types in both OCI and Direct modes
- Includes provider for UniDAC Express Edition
- Wide reporting component support, including support for InfoPower, ReportBuilder, FastReport
- Support for all standard visual data-aware controls
- Compatible with Delphi 6, 7, C++Builder 6, Borland Delphi Studio 2006, Code Gear RAD Studio 2007, 2009, Embarcadero RAD Studio 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, Seattle, Berlin, Tokyo, Rio, Sydney, Athens
- Support for Lazarus 3.2.0.0 and FPC 3.2.2 for Windows, macOS, and Linux (32-bit and 64-bit)
- Allows you to use Professional Edition of Delphi and C++Builder to develop client/server applications.

Oracle technology support:

- Oracle Advanced Queuing support
- Distributed transactions support with TOraTransaction component
- Oracle package support
- Support for Oracle alerts and pipes with TOraAlerter component
- Support for Direct Path interface with TOraLoader component
- Support for DBMS_TRACE package and SQLTrace functionality with TOraTrace
- Support for Oracle Change notifications functionality of Oracle 10g with TOraChangeNotification component
- Oracle Transparent Application Failover support
- Oracle 9i scrollable cursor support

- Multiple Oracle Homes support
- Oracle sequence support
- DML array operations support
- Direct lob access support
- Temporary LOB management routines
- Temporary LOBs for updating LOB fields
- OCI Connection Pooling , Proxy Session Pooling , and Statement Caching
- Oracle optimizer control
- ProxySession support
- External Procedure support
- CLIENT_IDENTIFIER support
- Statement Caching
- ROWID values retrieval
- Overloaded stored procedures support
- Support for WITH FUNCTION clause

Oracle data types:

- All Oracle data types support
- Oracle Object (including NOT FINAL objects) types support
- Nested table support
- PL/SQL table support
- PL/SQL records support
- Support for REF CURSORs
- XMLTYPE datatype support
- Oracle 9i TIMESTAMP and INTERVAL data types support

Performance:

- High overall performance

- Fast controlled fetch of large data blocks
- Optimized string data storing
- Advanced connection pooling
- High performance applying of cached updates with batches
- Caching of calculated and lookup fields
- Expanded fields in TSmartQuery
- Fast Locate in a sorted DataSet
- Preparation of user-defined update statements
- High performance batch processing
- Intelligent fetch block size control
- Advanced connection pooling
- SmartFetch Mode enabling fast bi-directional navigation through large datasets

Local data storage operations:

- Database-independent data storage with TVirtualTable component
- CachedUpdates operation mode
- Local sorting and filtering, with included calculated and lookup fields
- Local master/detail relationship
- Master/detail relationship in CachedUpdates mode

Data access and data management automation:

- Automatic data updating with TOraQuery , TSmartQuery and TOraTable components
- Automatic record refreshing and locking
- Automatic query preparing
- SmartRefresh option allows you to synchronize two or more datasets automatically
- Support for ftWideMemo field type in Delphi 2006 and higher
- Data Type Mapping
- Support for Data Encryption in a client application

Extended data access functionality:

- Separate component for executing SQL and PL/SQL blocks
- Simplified access to table data with TOraTable component
- BLOB compression support
- Support for using macros in SQL
- NonBlocking mode allows background executing and fetching data in separate threads
- Ability to customize update commands by attaching external components to TOraUpdateSQL objects
- Deferred detail DataSet refresh in master/detail relationships
- LargeInt fields support
- MIDAS technology support
- Ability to customize Oracle error messages with TOraErrorHandler
- Structural representation and editing of Oracle objects
- Fill DataSet with several REF CURSOR
- Fill DataSet with object, array and nested table data
- Object-oriented building of SELECT statements

Data exchange:

- Transferring data between all types of TDataSet descendants with TCRBatchMove component
- Data export and import to/from XML (ADO format)
- Ability to synchronize positions in different DataSets

Script execution:

- Advanced script execution features with TOraScript component
- Support for executing individual statements in scripts
- Support for executing huge scripts stored in files with dynamic loading
- Ability to use standard SQL*Plus tool syntax in scripts

SQL execution monitoring:

- Extended SQL tracing capabilities provided by TOraSQLMonitor component and DBMonitor
- Borland SQL Monitor support
- Ability to send messages to DBMonitor from any point in your program
- Display executing statement, all its parameters' values, and the type of parameters.

Visual extensions:

- Includes source code of enhanced TCRDBGrid data-aware grid control
- Customizable connection dialog
- Cursor changes during non-blocking execution

Design-time enhancements:

- DataSet Manager tool to control DataSet instances in the project
- Oracle Package Wizard that simplifies working with PL/SQL Packages
- Advanced design-time component and property editors
- Automatic design-time component linking
- Easy migration from BDE with Migration Wizard
- More convenient data source setup with the TOraDataSource component
- Syntax highlighting in design-time editors

Resources:

- Code documentation and guides in the CHM, PDF, and HXS formats
- Many helpful demo projects

Licensing and support:

- Included annual ODAC Subscription with Priority Support
- Licensed royalty-free per developer, per team, or per site

Request Support DAC Forum Provide Feedback

2.3 Requirements

Two versions of ODAC cannot be installed in parallel for the same IDE, and, since the Devart Data Access Components products have some shared bpl files, newer versions of ODAC may be incompatible with older versions of other Devart Data Access Components products.

So, before installing a new version of ODAC, uninstall any previous version of ODAC you may have, and check if your new install is compatible with other Devart Data Access Components products you have installed. For more information please see Using several products in one IDE. If you run into problems or have any compatibility questions, please email odac@devart.com

Note: You can avoid performing ODAC uninstallation manually when upgrading to a new version by directing the ODAC installation program to overwrite previous versions. To do this, execute the installation program from the command line with a /force parameter (Start | Run and type odacXX.exe /force, specifying the full path to the appropriate version of the installation program).

When installing ODAC from the sources to Windows Vista or Windows 7, it is necessary to have full access to the ODAC folder.

To use full set of Oracle features you have to have Oracle client software installed. If you do not want to install it, you can use Direct mode, in which ODAC communicates with Oracle server without intermediate libraries. In order to use the Direct mode, the operating system must have TCP/IP protocol support installed. For more information about using Direct mode, refer to Connecting in Direct mode article.

ODAC supports both 32-bit and 64-bit Oracle client versions in OCI mode. Developing applications for 64-bit client is possible only in RadStudio XE2 and Lazarus (FPC).

Note: RadStudio XE2 is a 32-bit application, therefore, for connecting to Oracle in OCI mode, even on a 64-bit platform, the 32-bit Oracle client is needed to be installed.

ODAC supports work with Oracle Instant Client in OCI mode. For correct work with Instant Client, the data about the client must be recorded to the registry or the Path environment variable, or Instant Client files(including tnsnames.ora) must be located in the application directory.

2.4 Compatibility

Oracle Compatibility

ODAC supports Oracle servers 23c, 21c, 19c, 18c, 12c, 11g, 10g, 9i, 8i, 8.0, including Oracle Express Edition 11g and 10g.

ODAC supports both x86 and x64 versions of the following Oracle Clients: 23c, 21c, 19c, 18c, 12c, 11g, 10g, 9i, 8i, 8.0.

Note that support for x64 versions of Oracle Clients is available in Delphi XE2 for 64-bit Windows and is not available in C++Builder and older versions of Delphi.

IDE Compatibility

ODAC is compatible with the following IDEs:

Embarcadero RAD Studio 12.1 Athens
- Embarcadero Delphi 12.1 Athens for Windows
- Embarcadero Delphi 12.1 Athens for macOS
- Embarcadero Delphi 12.1 Athens for Linux
- Embarcadero Delphi 12.1 Athens for iOS
- Embarcadero Delphi 12.1 Athens for Android
- Embarcadero C++Builder 12.1 Athens for Windows
- Embarcadero C++Builder 12.1 Athens for iOS
- Embarcadero C++Builder 12.1 Athens for Android

Embarcadero RAD Studio 12 Athens
- Embarcadero Delphi 12 Athens for Windows
- Embarcadero Delphi 12 Athens for macOS
- Embarcadero Delphi 12 Athens for Linux
- Embarcadero Delphi 12 Athens for iOS
- Embarcadero Delphi 12 Athens for Android
- Embarcadero C++Builder 12 Athens for Windows
- Embarcadero C++Builder 12 Athens for iOS

- Embarcadero C++Builder 12 Athens for Android

Embarcadero RAD Studio 11.1 Alexandria
- Embarcadero Delphi 11.1 Alexandria for Windows
- Embarcadero Delphi 11.1 Alexandria for macOS
- Embarcadero Delphi 11.1 Alexandria for Linux
- Embarcadero Delphi 11.1 Alexandria for iOS
- Embarcadero Delphi 11.1 Alexandria for Android
- Embarcadero C++Builder 11.1 Alexandria for Windows
- Embarcadero C++Builder 11.1 Alexandria for iOS
- Embarcadero C++Builder 11.1 Alexandria for Android

Embarcadero RAD Studio 10.4 Sydney (Requires Release 1 or Release 2)
- Embarcadero Delphi 10.4 Sydney for Windows
- Embarcadero Delphi 10.4 Sydney for macOS
- Embarcadero Delphi 10.4 Sydney for Linux
- Embarcadero Delphi 10.4 Sydney for iOS
- Embarcadero Delphi 10.4 Sydney for Android
- Embarcadero C++Builder 10.4 Sydney for Windows
- Embarcadero C++Builder 10.4 Sydney for iOS
- Embarcadero C++Builder 10.4 Sydney for Android

Embarcadero RAD Studio 10.3 Rio (Requires Release 2 or Release 3)
- Embarcadero Delphi 10.3 Rio for Windows
- Embarcadero Delphi 10.3 Rio for macOS
- Embarcadero Delphi 10.3 Rio for Linux
- Embarcadero Delphi 10.3 Rio for iOS
- Embarcadero Delphi 10.3 Rio for Android
- Embarcadero C++Builder 10.3 Rio for Windows
- Embarcadero C++Builder 10.3 Rio for macOS
- Embarcadero C++Builder 10.3 Rio for iOS

- Embarcadero C++Builder 10.3 Rio for Android

Embarcadero RAD Studio 10.2 Tokyo (Incompatible with Release 1)
- Embarcadero Delphi 10.2 Tokyo for Windows
- Embarcadero Delphi 10.2 Tokyo for macOS
- Embarcadero Delphi 10.2 Tokyo for Linux
- Embarcadero Delphi 10.2 Tokyo for iOS
- Embarcadero Delphi 10.2 Tokyo for Android
- Embarcadero C++Builder 10.2 Tokyo for Windows
- Embarcadero C++Builder 10.2 Tokyo for macOS
- Embarcadero C++Builder 10.2 Tokyo for iOS
- Embarcadero C++Builder 10.2 Tokyo for Android

Embarcadero RAD Studio 10.1 Berlin
- Embarcadero Delphi 10.1 Berlin for Windows
- Embarcadero Delphi 10.1 Berlin for macOS
- Embarcadero Delphi 10.1 Berlin for iOS
- Embarcadero Delphi 10.1 Berlin for Android
- Embarcadero C++Builder 10.1 Berlin for Windows
- Embarcadero C++Builder 10.1 Berlin for macOS
- Embarcadero C++Builder 10.1 Berlin for iOS
- Embarcadero C++Builder 10.1 Berlin for Android

Embarcadero RAD Studio 10 Seattle
- Embarcadero Delphi 10 Seattle for Windows
- Embarcadero Delphi 10 Seattle for macOS
- Embarcadero Delphi 10 Seattle for iOS
- Embarcadero Delphi 10 Seattle for Android
- Embarcadero C++Builder 10 Seattle for Windows
- Embarcadero C++Builder 10 Seattle for macOS
- Embarcadero C++Builder 10 Seattle for iOS

- Embarcadero C++Builder 10 Seattle for Android

Embarcadero RAD Studio XE8
- Embarcadero Delphi XE8 for Windows
- Embarcadero Delphi XE8 for macOS
- Embarcadero Delphi XE8 for iOS
- Embarcadero Delphi XE8 for Android
- Embarcadero C++Builder XE8 for Windows
- Embarcadero C++Builder XE8 for macOS
- Embarcadero C++Builder XE8 for iOS
- Embarcadero C++Builder XE8 for Android

Embarcadero RAD Studio XE7
- Embarcadero Delphi XE7 for Windows
- Embarcadero Delphi XE7 for macOS
- Embarcadero Delphi XE7 for iOS
- Embarcadero Delphi XE7 for Android
- Embarcadero C++Builder XE7 for Windows
- Embarcadero C++Builder XE7 for macOS
- Embarcadero C++Builder XE7 for iOS
- Embarcadero C++Builder XE7 for Android

Embarcadero RAD Studio XE6
- Embarcadero Delphi XE6 for Windows
- Embarcadero Delphi XE6 for macOS
- Embarcadero Delphi XE6 for iOS
- Embarcadero Delphi XE6 for Android
- Embarcadero C++Builder XE6 for Windows
- Embarcadero C++Builder XE6 for macOS
- Embarcadero C++Builder XE6 for iOS
- Embarcadero C++Builder XE6 for Android

Embarcadero RAD Studio XE5 (Requires Update 2)
- Embarcadero Delphi XE5 for Windows
- Embarcadero Delphi XE5 for macOS
- Embarcadero Delphi XE5 for iOS
- Embarcadero Delphi XE5 for Android
- Embarcadero C++Builder XE5 for Windows
- Embarcadero C++Builder XE5 for macOS
- Embarcadero C++Builder XE5 for iOS

Embarcadero RAD Studio XE4
- Embarcadero Delphi XE4 for Windows
- Embarcadero Delphi XE4 for macOS
- Embarcadero Delphi XE4 for iOS
- Embarcadero C++Builder XE4 for Windows
- Embarcadero C++Builder XE4 for macOS

Embarcadero RAD Studio XE3 (Requires Update 2)
- Embarcadero Delphi XE3 for Windows
- Embarcadero Delphi XE3 for macOS
- Embarcadero C++Builder XE3 for Windows
- Embarcadero C++Builder XE3 for macOS

Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
- Embarcadero Delphi XE2 for Windows
- Embarcadero Delphi XE2 for macOS
- Embarcadero C++Builder XE2 for Windows
- Embarcadero C++Builder XE2 for macOS

Embarcadero RAD Studio XE
- Embarcadero Delphi XE
- Embarcadero C++Builder XE

Embarcadero RAD Studio 2010
- Embarcadero Delphi 2010

- Embarcadero C++Builder 2010

CodeGear RAD Studio 2009 (Requires Update 3)
- CodeGear Delphi 2009
- CodeGear C++Builder 2009

CodeGear RAD Studio 2007
- CodeGear Delphi 2007
- CodeGear C++Builder 2007

Borland Developer Studio 2006
- Borland Delphi 2006
- Borland C++Builder 2006

Borland Delphi 7
Borland Delphi 6 (Requires Update Pack 2 – Delphi 6 Build 6.240)
Borland C++Builder 6 (Requires Update Pack 4 – C++Builder 6 Build 10.166)
Lazarus 3.2.0 and Free Pascal 3.2.2 for Windows, macOS, and Linux.

All the existing Delphi and C++Builder editions are supported: Architect, Enterprise, Professional, Community, and Starter.

Lazarus and Free Pascal are supported only in Trial Edition and Professional Edition with source code.

Supported Target Platforms

- Windows 32-bit and 64-bit
- macOS 64-bit and ARM (Apple Silicon M1)
- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit
- iOS 64-bit
- iOS Simulator ARM 64-bit
- Android 32-bit and 64-bit

Support for Windows 64-bit is available since RAD Studio XE2. Support for iOS 64-bit is available since RAD Studio XE8. Support for Android 32-bit is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since

RAD Studio 10.3.3 Rio.

Supported GUI Frameworks

- FireMonkey (FMX)
- Visual Component Library (VCL)
- Lazarus Component Library (LCL)

Devart Data Access Components Compatibility

All DAC products are compatible with each other.

But, to install several DAC products to the same IDE, it is necessary to make sure that all DAC products have the same common engine (BPL files) version. The latest versions of DAC products or versions with the same release date always have the same version of the common engine and can be installed to the same IDE.

 Request Support DAC Forum Provide Feedback

2.5 Using Several DAC Products in One IDE

UniDAC, ODAC, SDAC, MyDAC, IBDAC, PgDAC, LiteDAC and VirtualDAC components use common base packages listed below:

Packages:

- dacXX.bpl
- dacvclXX.bpl
- dcldacXX.bpl

Note that product compatibility is provided for the current build only. In other words, if you upgrade one of the installed products, it may conflict with older builds of other products. In order to continue using the products simultaneously, you should upgrade all of them at the same time.

 Request Support DAC Forum Provide Feedback

2.6 Component List

This topic presents a brief description of the components included in the Oracle Data Access Components library. Click on the name of each component for more information. These components are added to the ODAC page of the Component palette except for TCRBatchMove and TVirtualTable components. TCRBatchMove and TVirtualTable components are added to the Data Access page of the Component palette. Basic ODAC components are included in all ODAC editions. ODAC Professional and Developer Edition components are not included in ODAC Standard Edition.

Basic ODAC components

	TOraSession	Lets you set up and control connections to Oracle.
	TOraQuery	Uses SQL statements to retrieve data from Oracle tables and pass it to one or more data-aware components through a TDataSource object. This component provides a mechanism for updating data in Oracle.
	TSmartQuery	A more efficient query component. This component is aware of all the fields that belong to a table being updated and performs on-demand full row retrieval with Expand Fields . A traffic-efficient alternative to TOraQuery for working with large tables with lots of fields. Includes Smart Refresh functionality in Professional and Developer Editions.
	TOraSQL	Executes SQL statements, PL/SQL blocks, and stored procedures, which do not return rowsets.
	TOraTable	Lets you retrieve and update data in a single table without writing SQL statements.
	TOraStoredProc	Executes stored procedures and functions. Lets you edit cursor data returned as parameter.
	TOraNestedTable	Component for controlling nested table data.
	TOraUpdateSQL	Lets you tune update operations for a DataSet component.
	TOraDataSource	Provides an interface for connecting data-aware controls on a form and ODAC dataset components.

	TOraScript	Executes sequences of SQL and PL/SQL statements.
	TOraSQLMonitor	Interface for monitoring dynamic SQL execution in ODAC-based applications.
	TConnectDialog	Allows you to build custom prompts for usernames, passwords, and server names.
	TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.
	TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

ODAC Professional and Developer Edition components

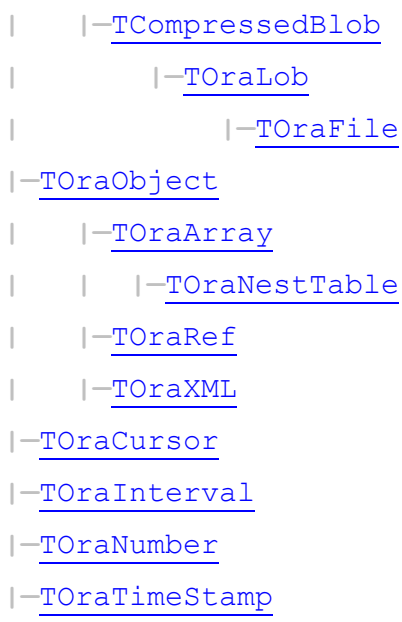
	TOraEncryptor	Represents data encryption and decryption in client application.
	TOraPackage	Provides a straightforward way to access Oracle packages.
	TOraAlerter	Allows you to transfer messages between sessions.
	TOraLoader	Allows you to quickly load data into Oracle databases.
	TOraTransaction	Provides discrete transaction control over sessions. Can be used to manipulate both simple and distributed transactions.
	TOraQueue	Lets you monitor the message queue. Provides an interface for enqueuing and dequeuing messages.
	TOraQueueTable	Component for managing queue tables.
	TOraQueueAdmin	Component for managing queues.
	TOraChangeNotification	Allows you to use Oracle Database Change Notifications.
	TOraTrace	Allows you to start and stop the SQL trace for a specified session. This component provides access to the DBMS_TRACE package.

	TOraErrorHandler	Translates error messages.
	TOraProvider	Loads data to and from a dataset.
	TOraMetaData	Retrieves metadata on specified SQL object.
	TCRBatchMove	Transfers data between all types of TDataSet descendants. This component is placed on the Data Access page of the Component palette.

2.7 Hierarchy Chart

Many ODAC classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for ODAC is shown below. The ODAC classes are represented by hyperlinks that point to their description in this documentation. The description of the standard classes can be found in the documentation of your IDE.

```
TObject
  |—TPersistent
      |—TComponent
      |    |—TCustomConnection
      |    |    |—TCustomDAConnection
      |    |         |—TOraSession
      |    |—TDataSet
      |    |    |—TMemDataSet
      |    |         |—TCustomDADataSet
      |    |         |    `|—TOraDataSet
      |    |         |         |—TCustomOraQuery
      |    |         |              |—TCustomSmartQuery
      |    |         |              |    |—TOraTable
      |    |         |              |    |—TSmartQuery
      |    |         |              |—TOraQuery
```



 Request Support DAC Forum Provide Feedback

2.8 Editions

Oracle Data Access Components comes in two editions: Standard and Professional.

The **Standard** edition includes the ODAC basic connectivity components and ODAC Migration Wizard. ODAC Standard Edition is a cost-effective solution for database application developers who are looking for high-performance connectivity to Oracle for secure, reliable, and high-speed data transmission.

The **Professional** edition shows off the full power of ODAC, enhancing ODAC Standard Edition with support for Oracle-specific functionality, access to the Direct mode for connecting to the Oracle server directly via TCP/IP, and some advanced dataset management features.

You can get **Source Access** to the Client mode implementation of all the component classes in ODAC by purchasing a special ODAC Professional Edition with Source Code. The source code of DataSet Manager and Migration Wizard is not distributed. The source code of the Direct mode for Oracle is distributed obfuscated.

The matrix below compares the features of ODAC editions. See Features for the detailed list of ODAC features.

ODAC Edition Matrix

Feature	Standard	Professional
Direct Connectivity		
Connection without Oracle client	✗	✓
Desktop Application Development		
Windows	✓	✓
macOS	✗	✓
Linux	✗	✓
Mobile Application Development		
iOS	✗	✓
Android	✗	✓
Data Access Components		
TOraSession	✓	✓
TOraQuery	✓	✓
TOraTable	✓	✓
TOraStoredProc	✓	✓
TOraUpdateSQL	✓	✓
TOraSQL	✓	✓
TOraDataSource	✓	✓
Script executing TOraScript	✓	✓
Transactions managing TOraTransaction	✗	✓
Fast data loading into the server TOraLoader	✗	✓

Advanced Query Components		
Expanded field representation TSmartQuery	✓	✓
Smart refresh in TSmartQuery component	✗	✓
Oracle Specific Components		
Oracle packages TOraPackage	✗	✓
Oracle nested tables TOraNestedTable	✓	✓
Messaging between sessions and applications TOraAlerter	✗	✓
Reaction on server side changes on-the-fly TOraChangeNotification	✗	✓
Oracle advanced queing TOraQueue TOraQueueAdmin TOraQueueTable	✗	✓
PL/SQL tracing TOraTrace	✗	✓
Obtaining metadata about database objects TOraMetaData	✗	✓
Oracle errors handling TOraErrorHandler	✗	✓
DataBase Activity Monitoring		
Monitoring of per-component SQL execution TOraSQLMonitor	✓	✓
Additional Components		
Advanced connection dialog TConnectDialog	✗	✓
Data encryption and decryption TOraEncryptor	✗	✓
Advanced DataSet provider TOraProvider	✗	✓
Data storing in memory table TVirtualTable	✓	✓
Dataset that wraps arbitrary non-tabular data TVirtualDataSet	✓	✓

Advanced DBGrid with extended functionality TCRDBGrid	✓	✓
Records transferring between datasets TCRBatchMove	✗	✓
Design-Time Features		
Enhanced component and property editors	✓	✓
Migration Wizard	✓	✓
DataSet Manager	✗	✓
Oracle Package Wizard	✗	✓
Cross IDE Support		
Lazarus and Free Pascal Support	✗	SRC[1]

[1] Available only in Professional Edition with Source Code.

Request Support DAC Forum Provide Feedback

2.9 Licensing

PLEASE READ THIS LICENSE AGREEMENT CAREFULLY. BY INSTALLING OR USING THIS SOFTWARE, YOU INDICATE ACCEPTANCE OF AND AGREE TO BECOME BOUND BY THE TERMS AND CONDITIONS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT INSTALL OR USE THIS SOFTWARE AND PROMPTLY RETURN IT TO DEVART.

INTRODUCTION

This Devart end-user license agreement ("Agreement") is a legal agreement between you (either an individual person or a single legal entity) and Devart, for the use of ODAC software application, source code, demos, intermediate files, printed materials, and online or electronic documentation contained in this installation file. For the purpose of this Agreement, the software program(s) and supporting documentation will be referred to as the "Software".

LICENSE

1. GRANT OF LICENSE

The enclosed Software is licensed, not sold. You have the following rights and privileges, subject to all limitations, restrictions, and policies specified in this Agreement.

1.1. If you are a legally licensed user, depending on the license type specified in the registration letter you have received from Devart upon purchase of the Software, you are entitled to either:

- install and use the Software on one or more computers, provided it is used by 1 (one) for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Single Developer License"); or
- install and use the Software on one or more computers, provided it is used by up to 4 (four) developers within a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Team Developer License"); or
- install and use the Software on one or more computers, provided it is used by developers in a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Site License").

1.2. If you are a legally licensed user of the Software, you are also entitled to:

- make one copy of the Software for archival purposes only, or copy the Software onto the hard disk of your computer and retain the original for archival purposes;
- develop and test applications with the Software, subject to the Limitations below;
- create libraries, components, and frameworks derived from the Software for personal use only;
- deploy and register run-time libraries and packages of the Software, subject to the Redistribution policy defined below.

1.3. You are allowed to use evaluation versions of the Software as specified in the Evaluation section.

No other rights or privileges are granted in this Agreement.

2. LIMITATIONS

Only legally registered users are licensed to use the Software, subject to all of the conditions of this Agreement. Usage of the Software is subject to the following restrictions.

2.1. You may not reverse engineer, decompile, or disassemble the Software.

2.2. You may not build any other components through inheritance for public distribution or commercial sale.

2.3. You may not use any part of the source code of the Software (original or modified) to build any other components for public distribution or commercial sale.

2.4. You may not reproduce or distribute any Software documentation without express written permission from Devart.

2.5. You may not distribute and sell any portion of the Software without integrating it into your Applications as Executable Code, except a Trial version that can be distributed for free as original Devart's ODAC Trial package.

2.6. You may not transfer, assign, or modify the Software in whole or in part. In particular, the Software license is non-transferable, and you may not transfer the Software installation package.

2.7. You may not remove or alter any Devart's copyright, trademark, or other proprietary rights notice contained in any portion of Devart units, source code, or other files that bear such a notice.

3. REDISTRIBUTION

The license grants you a non-exclusive right to compile, reproduce, and distribute any new software programs created using ODAC. You can distribute ODAC only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.

All Devart's units, source code, and other files remain Devart's exclusive property.

4. TRANSFER

You may not transfer the Software to any individual or entity without express written permission from Devart. In particular, you may not share copies of the Software under “Single Developer License” and “Team License” with other co-developers without obtaining proper

license of these copies for each individual.

5. TERMINATION

Devart may immediately terminate this Agreement without notice or judicial resolution in the event of any failure to comply with any provision of this Agreement. Upon such termination you must destroy the Software, all accompanying written materials, and all copies.

6. EVALUATION

Devart may provide evaluation ("Trial") versions of the Software. You may transfer or distribute Trial versions of the Software as an original installation package only. If the Software you have obtained is marked as a "Trial" version, you may install and use the Software for a period of up to 60 calendar days from the date of installation (the "Trial Period"), subject to the additional restriction that it is used solely for evaluation of the Software and not in conjunction with the development or deployment of any application in production. You may not use applications developed using Trial versions of the Software for any commercial purposes. Upon expiration of the Trial Period, the Software must be uninstalled, all its copies and all accompanying written materials must be destroyed.

7. WARRANTY

The Software and documentation are provided "AS IS" without warranty of any kind. Devart makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or use.

8. SUBSCRIPTION AND SUPPORT

The Software is sold on a subscription basis. The Software subscription entitles you to download improvements and enhancement from Devart’s web site as they become available, during the active subscription period. The initial subscription period is one year from the date of purchase of the license. The subscription is automatically activated upon purchase, and may be subsequently renewed by Devart, subject to receipt applicable fees. Licensed users of the Software with an active subscription may request technical assistance with using the Software over email from the Software development. Devart shall use its reasonable endeavours to answer queries raised, but does not guarantee that your queries or problems will be fixed or solved.

Devart reserves the right to cease offering and providing support for legacy IDE versions.

9. COPYRIGHT

The Software is confidential and proprietary copyrighted work of Devart and is protected by international copyright laws and treaty provisions. You may not remove the copyright notice from any copy of the Software or any copy of the written materials, accompanying the Software.

This Agreement contains the total agreement between the two parties and supersedes any other agreements, written, oral, expressed, or implied.

Request Support DAC Forum Provide Feedback

2.10 Getting Support

This page lists several ways you can find help with using ODAC and describes the ODAC Priority Support program.

Support Options

There are a number of resources for finding help on installing and using ODAC.

- You can find out more about ODAC installation or licensing by consulting the Licensing and FAQ sections.
- You can get community assistance and technical support on the ODAC Community Forum.
- You can get advanced technical assistance by ODAC developers through the **ODAC Priority Support** program.

If you have a question about ordering ODAC or any other Devart product, please contact sales@devart.com.

ODAC Priority Support

ODAC Priority Support is an advanced product support service for getting expedited individual assistance with ODAC-related questions from the ODAC developers themselves. Priority Support is carried out over email and has two business days response policy. Priority Support is available for users with an active ODAC Subscription.

To get help through the ODAC Priority Support program, please send an email to

support@devart.com describing the problem you are having. Make sure to include the following information in your message:

- The version of Delphi, C++Builder you are using.
- Your ODAC Registration number.
- Full ODAC edition name and version number. You can find both of these from the ODAC | ODAC About menu in the IDE.
- Versions of the Oracle server and client you are using.
- A detailed problem description.
- If possible, a small test project that reproduces the problem. It is recommended to use Scott or SYS schema objects only. Please include definitions for all and avoid using third-party components.

Request Support DAC Forum Provide Feedback

2.11 Frequently Asked Questions

This page contains a list of Frequently Asked Questions for Oracle Data Access Components.

If you have encounter a question with using ODAC, please browse through this list first. If this page does not answer your question, refer to the Getting Support topic in ODAC help.

Installation and Deployment

1. I'm having a problem with installing ODAC or compiling ODAC-based projects...

You may be having a compatibility issue that shows up in one or more of the following forms:

- Get a "Setup has detected already installed DAC packages which are incompatible with current version" message during ODAC installation.
- Get a "Procedure entry point ... not found in ... " message when starting IDE.
- Get a "Unit ... was compiled with a different version of ..." message on compilation.

You can have such problems if you installed incompatible ODAC, SDAC, MyDAC or IBDAC

versions. All these products use common base packages. The easiest way to avoid the problem is to uninstall all installed DAC products and then download from our site and install the last builds.

2. What software should be installed on a client computer so that my applications that use ODAC can run?

To use the full set of Oracle features, the client computer has to have Oracle client software (OCI) installed. If you do not want to install OCI, you can use Direct mode, in which ODAC communicates with Oracle server without intermediate libraries. In order to use the Direct mode, the operating system on the client computer must have TCP/IP protocol support installed.

3. How can I quickly convert a project from BDE to ODAC?

To quickly migrate your project from BDE you can use the BDE Migration Wizard. To start it, open your project and choose BDE Migration Wizard from the ODAC menu of your IDE.

Licensing and Subscriptions

1. Am I entitled to distribute applications written with ODAC?

If you have purchased a full version of ODAC, you are entitled to distribute pre-compiled programs created with its use. You are not entitled to propagate any components inherited from ODAC or using ODAC source code. For more information see the *License.rtf* file in your ODAC installation directory.

2. Can I create components using ODAC?

You can create your own components that are inherited from ODAC or that use the ODAC source code. You are entitled to sell and distribute compiled application executables that use such components, but not their source code and not the components themselves.

3. What licensing changes can I expect with ODAC 6.00?

The basic ODAC license agreement will remain the same. With ODAC 6.00, the ODAC Edition Matrix will be reorganized and a new ODAC Subscription Program will be introduced.

4. What do the ODAC 6.00 Edition Levels correspond to?

ODAC 6.00 will come in six editions: Trial, Standard, Professional, Professional with Sources,

Developer, and Developer with Sources.

When you upgrade to the new version, your edition level will be automatically updated using the following Edition Correspondence Table.

Edition Correspondence Table for Upgrading to ODAC 6.00

Old Edition Level	New Edition Level
- No Correspondence -	ODAC Standard Edition
ODAC Standard Edition	ODAC Professional Edition
ODAC Net Edition	ODAC Professional Edition
ODAC Professional Edition	ODAC Professional Edition with Sources
- No Correspondence -	ODAC Developer Edition
- No Correspondence -	ODAC Developer Edition with Sources
ODAC Trial Edition	ODAC Trial Edition

The feature list for each edition can be found in the ODAC documentation and the ODAC website.

5. I have a registered version of ODAC. Will I need to pay to upgrade to future versions?

After ODAC 6.00, all upgrades to future versions are free to users with an active ODAC Subscription.

Users that have a registration for versions of ODAC prior to ODAC 6.00 will have to first upgrade to ODAC 6.00 to jump in on the Subscription Program.

6. What are the benefits of the ODAC Subscription Program?

The ODAC **ODAC Subscription Program** is an annual maintenance and support service for

ODAC users.

Users with a valid ODAC Subscription get the following benefits:

- Access to new versions of ODAC when they are released
- Access to all ODAC updates and bug fixes
- Product support through the ODAC Priority Support program
- Notification of new product versions

Priority Support is an advanced product support program which offers you expedited individual assistance with ODAC-related questions from the ODAC developers themselves. Priority Support is carried out over email and has a guaranteed two business day response policy.

The ODAC Subscription Program is available for registered users of ODAC 6.00 and higher.

7. Can I use my version of ODAC after my Subscription expires?

Yes, you can. ODAC version licenses are perpetual.

8. I want a ODAC Subscription! How can I get one?

You can renew your ODAC Subscription on the ODAC Ordering Page. For more information, please contact sales@devart.com.

You will be able to renew your ODAC Subscription by email or on the ODAC website. For more information, please contact sales@deavrt.com.

9. Does this mean that if I upgrade to ODAC 6 from ODAC 5, I'll get an annual ODAC Subscription for free?

Yes.

10.How do I upgrade to ODAC 6.00?

To upgrade to ODAC 6.00, you can get a Version Update from the ODAC Ordering Page. For more information, please contact sales@devart.com.

Performance

1. How productive is ODAC?

From time to time we compare ODAC with other products, and ODAC always takes first place. For more information, please refer to the ODAC performance comparison results posted on the ODAC website

2. Why does the Locate function work so slowly the first time I use it?

Locate is performed on the client. So if you had set FetchAll to False when opening your dataset, cached only some of the rows on the client, and then invoked Locate, ODAC will have to fetch all the remaining rows from the server before performing the operation. On subsequent calls, Locate should work much faster.

If the Locate method keeps working slowly on subsequent calls or if you are working with FetchAll=True, try the following. Perform local sorting by a field that is used in the Locate method. Just assign corresponding field name to the IndexFieldNames property.

How To

1. How can I find out which version of ODAC I am using ?

You can determine your ODAC version number in several ways:

- During installation of ODAC, consult the ODAC Installer screen.
- After installation, see the *history.html* file in your ODAC installation directiory.
- At design-time, select Oracle | About ODAC from the main menu of your IDE.
- At run-time, check the value of the OdacVersion and DACVersion constants.

2. How can I stop the cursor from changing to an hour glass during query execution?

Just set the DBAccess.ChangeCursor variable to False anywhere in your program. The cursor will stop changing after this command is executed.

3. How can I execute a query saved in the SQLInsert, SQLUpdate, SQLDelete, or SQLRefresh properties of a ODAC dataset?

The values of these properties are templates for query statements, and they cannot be manually executed. Usually there is no need to fill these properties because the text of the query is generated automatically.

In special cases, you can set these properties to perform more complicated processing during a query. These properties are automatically processed by ODAC during the execution

of the Post, Delete, or RefreshRecord methods, and are used to construct the query to the server. Their values can contain parameters with names of fields in the underlying data source, which will be later replaced by appropriate data values.

For example, you can use the SQLInsert template to insert a row into a query instance as follows.

- Fill the SQLInsert property with the parametrized query template you want to use.
- Call Insert.
- Initialize field values of the row to insert.
- Call Post.

The value of the SQLInsert property will then be used by ODAC to perform the last step.

Setting these properties is optional and allows you to automatically execute additional SQL statements, add calls to stored procedures and functions, check input parameters, and/or store comments during query execution. If these properties are not set, the ODAC dataset object will generate the query itself using the appropriate insert, update, delete, or refresh record syntax.

4. My program allows users to edit records directly in a DBGrid instance. How can I disable record deletion?

If TOraQuery acts as TDataSet, it is very simple to prohibit deleting, inserting and/or updating of records. Simply clear the relevant property (SQLDelete, SQLInsert,SQLUpdate). The action with empty SQL statement will not be allowed.

5. How do I allow users to delete, insert, and edit records (e.g. in DBGrid), but ensure deletions are not represented in the database?

Assign the following PL/SQL block to TOraQuery.SQLDelete:

```
begin
  Null;
end;
```

6. How can I tune the NUMBER fields definition in ODAC?

There are three typed constants in the OraClasses.pas module: IntegerPrecision, LargeIntPrecision and FloatPrecision. Using the values of these constants and the

EnableIntegers and EnableNumbers options, the Oracle NUMBER type is mapped to ODAC field classes as follows.

Conditions	Field class
Precision <= IntegerPrecision, Scale = 0, EnableIntegers = True	TIntegerField
IntegerPrecision < Precision <= LargeIntPrecision, Scale = 0, EnableIntegers = True	TLargeIntField
Precision > FloatPrecision, Scale > 0, EnableNumbers = True	TOraNumberField
In other cases	TFloatField

The default values of these constants are:

IntegerPrecision = 9, LargeIntPrecision = 0, FloatPrecision = 15

7. How to enable support of TLargeIntField?

Set the value of the global constant LargeIntPrecision to 18.

General Questions

1. What are the advantages of ODAC compared to BDE?

BDE provides a more or less uniform way for accessing different servers (SQL Server, MySQL, Oracle and so on)

ODAC is a set of components optimized for working specifically with Oracle, and has a server-specific component interface and advanced design-time support.

As a result, there are a number of reasons why ODAC may be better than BDE for your Oracle-based application. Some of them are enumerated here. For more information refer to the ODAC feature list.

- Incomparably higher speed of data access, fetching, and processing
- Possibility to individually adjust and optimize execution parameters for every query
- Complete support of PL/SQL

- Return of cursor from a stored procedure or anonymous PL/SQL block (by parameter)
- Return of PL/SQL result sets from a stored procedure or anonymous PL/SQL block
- Support for asynchronous execution of queries
- Possibility to break the execution of long-duration queries
- Built-in system for debugging in run-time
- Convenient and simple-to-use standardized error processing mechanism
- No need to install and configure BDE on client machines during deployment

2. What happened to the ODAC Net option and the TOraSession.Options.Net property?

As of ODAC 6.00, the ODAC Net Option has been renamed to **ODAC Direct mode**, and TOraSession.Options.Net has been replaced with TOraSession.Options.Direct. You can configure ODAC to connect directly to Oracle over TCP/IP by setting TOraSession.Options.Direct to True.

Note TOraSession.Options.Net has been retained for backwards-compatibility. This property is depreciated as of ODAC 6.00. Use TOraSession.Options.Direct instead.

3. Are the ODAC connection components thread-safe?

In Client mode, ODAC is thread-safe. In Direct mode, we do not guarantee complete thread safety and recommend setting up a separate Connection set for each thread that uses ODAC.

4. Behaviour of my application has changed when I upgraded ODAC. How can I restore the old behaviour with the new version?

We always try to keep ODAC compatible with previous versions, but sometimes we have to change behaviour of ODAC in order to enhance its functionality, or avoid bugs. If either of changes is undesirable for your application, and you want to save the old behaviour, please refer to the "Compatibility with previous versions" topic in ODAC help. This topic describes such changes, and how to revert to the old ODAC behaviour.

5. When editing a DataSet, I get an exception with the message 'Update failed. Found

%d records.' or 'Refresh failed. Found %d records.'

This error occurs when the database server is unable to determine which record to modify or delete. In other words, there are either more than one record or no records that suit the UPDATE criteria. Such situation can happen when you omit the unique field in a SELECT statement (TCustomDADataSet.SQL) or when another user modifies the table simultaneously. This exception can be suppressed. Refer to TCustomDADataSet.Options topic in ODAC help for more information.

6. I would like to use MIDAS technology. Does ODAC support the IProvider interface?

Yes. Check out the Provider property of the TOraProvider component.

7. What's the difference between TOraQuery, TSmartQuery, and TOraTable?

All these components are inherited from TDataSet and have all its capabilities. However, each component has a number of differences.

- TOraQuery represents the most general way of executing queries and editing data. It is the most universal component, while TSmartQuery and TOraTable are designed for convenience only.
- TSmartQuery includes all the functionality of TOraQuery component, and provides additional features: expand fields, that lets all data controls be aware of all the fields belonging to updating table an not only those requested in SELECT clause, and smart refresh (in Professional and Developer editions only).
- TOraTable makes it unnecessary to write SQL statements to perform both data updates and data selection and emulates working with a local table. With TOraTable, you do not have to use SQL at all, and only have to specify the name of the appropriate table or view.

8. Can ODAC and BDE functions be used side-by-side in a single application?

Yes. There is no problem with using both ODAC and BDE functions in the same application.

3 Getting Started

This section introduces Oracle Data Access Components. It contains the information on how to install Oracle Data Access Components, quick walktroughs to get started developing applications with it, information on technical licensing and deployment, and brief description of ODAC documentation and samples.

- Installation
- Migration Wizard
- Connecting to Oracle
- Creating Database Objects
- Retrieving and Modifying Data
- Inserting Data Into Tables
- Working With Oracle Stored Procedures
- Using Transactions

Request Support DAC Forum Provide Feedback

3.1 Installation

This topic contains the environment changes made by the ODAC installer. If you are having problems with using ODAC or compiling ODAC-based products, check this list to make sure that your system is properly configured.

Compiled versions of ODAC are installed automatically by the ODAC Installer for all supported IDEs except for Lazarus. Versions of ODAC with Source Code must be installed manually. Installation of ODAC from sources is described in the supplied *ReadmeSrc.html* file.

Table of Contents

- Additional packages for using ODAC managers and wizards

3. Environment Changes
   - Delphi
   - C++Builder
   - Lazarus
4. Installation of Additional Components and Add-ins
   - TOraProvider
   - DBMonitor
5. Uninstalling
   - Delphi and C++Builder
   - Lazarus

Before installing ODAC ...

Please read our Compatibility page and make sure that your system satisfies the requirements.

Two versions of ODAC cannot be simultaneously installed for the same IDE, and, since the Devart Data Access Components products have some shared bpl files, newer versions of ODAC may be incompatible with older versions of MyDAC, IBDAC, and SDAC.

So before installing a new version of ODAC, uninstall any previous version of ODAC you may have, and check if your new install is compatible with other Devart Data Access Components products you have installed. For more information please see Using several products in one IDE. If you run into problems or have any compatibility questions, please email odac@devart.com

Note: You can avoid performing ODAC uninstallation manually when upgrading to a new version by directing the ODAC installation program to overwrite previous versions. To do this, execute the installation program from the command line with a `/force` parameter (Start | Run and type `odacXX.exe /force`, specifying the full path to the appropriate version of the installation program).

Installed packages

Note: %ODAC% denotes the path to your ODAC installation directory.

Delphi/C++Builder Win32 project packages

Name	*Description*	*Location*
dacXX.bpl	DAC run-time package	Windows\System32
dcldacXX.bpl	DAC design-time package	Delphi\Bin
dacvclXX.bpl*	DAC VCL support package	Delphi\Bin
odacXX.bpl	ODAC run-time package	Windows\System32
dclodacXX.bpl	ODAC design-time package	Delphi\Bin
odacvclXX.bpl*	VCL support package	Delphi\Bin
oraprovXX.bpl	TOraProvider component	Delphi\Bin
crcontrolsXX.bpl	TCRDBGrid component	Delphi\Bin

Additional packages for using ODAC managers and wizards

Name	*Description*	*Location*
datasetmanagerXX.bpl	DataSet Manager package	Delphi\Bin
oramigwizardXX.dll	ODAC BDE Migration wizard	%ODAC%\Bin

Environment Changes

To compile ODAC-based applications, your environment must be configured to have access to the ODAC libraries. Environment changes are IDE-dependent.

For all instructions, replace %ODAC% with the path to your ODAC installation directory

Delphi

- %ODAC%\Lib should be included in the Library Path accessible from Tools | Enviroment options | Library.

The ODAC Installer performs Delphi environment changes automatically for compiled versions of ODAC.

C++Builder

C++Builder 6:

- `$(BCB)\ODAC\Lib` should be included in the Library Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.
- `$(BCB)\ODAC\Include` should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006 and higher:

- `$(BCB)\ODAC\Lib` should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.
- `$(BCB)\ODAC\Include` should be included in the Include search path of the Default Project Options accessible from Project | Default Options | C++Builder | C++ Compiler | Paths and Defines.

The ODAC Installer performs C++Builder environment changes for compiled versions of ODAC automatically.

Lazarus

The ODAC installation program only copies ODAC files. You need to install ODAC packages to Lazarus IDE manually. Open %ODAC%\Source\Lazarus1\dclodac10.lpk (for Trial version %ODAC%\Packages\dclodac10.lpk) file in Lazarus and press the Install button. After that Lazarus IDE will be rebuilded with ODAC packages.

Do not press the Compile button for the package. Compiling will fail because there are no ODAC sources.

To check that your environment has been properly configured, try to compile one of the demo projects included with ODAC. The ODAC demo projects are located in %ODAC%/Demos.

Installation of Additional Components and Add-ins

TOraProvider

If you use Delphi Enterprise Edition, Delphi Architect Edition, or C++Builder Enterprise Edition, you can install the TOraProvider component by compiling and installing the

oraprovXX.bpk package.

DBMonitor

DBMonitor is a an easy-to-use tool to provide visual monitoring of your database applications. It is provided as an alternative to Borland SQL Monitor which is also supported by ODAC. DBMonitor is intended to hamper an application that is being monitored as little as possible. For more information, visit the DBMonitor page online.

Uninstalling

To uninstall the Compiled versions of ODAC, select **Settings -> Control Panel -> Add or Remove Programs** from the Start menu (Windows XP and earlier) or select Control Panel from the Start menu and click **Uninstall a program** in the Control Panel window (Windows Vista and higher). Then select ODAC from the installed software list and click **Uninstall** (Windows XP and earlier) or **Uninstall/Change** (Windows Vista and higher).

Delphi and C++Builder

To uninstall Source versions of ODAC, select **Install Packages...** from the **Components** menu and remove the following packages:

- Devart Controls (CrControlsXXX.bpl)
- Devart Data Access Components (dcldavXXX.bpl)
- Devart Data Access GUI Related Components (dacvclXXX.bpl)
- Devart DataSet Manager (DataSetManagerXXX.bpl)
- Oracle Data Access Components (dclodacXXX.bpl)
- Oracle Data Access GUI Related Components (odacvclXXX.bpl)
- OraProvider Package (oraprovXXX.bpl)

Lazarus

To uninstall Source versions of ODAC, start the IDE, select **Install/Uninstall Packages** from the **Packages** menu, and remove the following packages.

- dac10 xx.xx.xx.xx
- dacvcl10 xx.xx.xx.xx

- dcldac10 xx.xx.xx.xx
- dclodac10 xx.xx.xx.xx
- odac10 xx.xx.xx.xx
- odacvxl10 xx.xx.xx.xx

After removing packages, you must also remove all the *dac*.* files from the Windows \System32, Delphi\Bin and delete the %ODAC% folder.

Request Support DAC Forum Provide Feedback

3.2 Migrating from BDE and DOA

Note: Migration Wizard is only available for Delphi.

Migration Wizard allows you to convert your BDE or DOA projects to ODAC. This wizard replaces BDE or DOA components in a specified project (.dfm and .pas files) with ODAC components.

To convert a project, perform the following steps.

- Select **Migration Wizard** from the **ODAC** menu
- Select **Replace BDE components** to replace the corresponding components with ODAC ones and click the Next button. If you need to replace **DOA components**, select **DOA** from the drop-down list on the right and click Next.
- Select the location of the files to search - current open project or disc folder.
- If you have selected Disc folder on the previous step, specify the required folder and specify whether to process subfolders. Press the Next button.
- Select whether to make backup (it is highly recommended to make a backup), backup location, and log parameters, and press the Next button. Default backup location is RBackup folder in your project folder.
- Check your settings and press the Finish button to start the conversion operation.
- The project should be saved before conversion. You will be asked before saving it. Click Yes to continue project conversion.

After the project conversion it will be reopened.

The Wizard just replaces all standard BDE components. Probably you will need to make some changes manually to compile your application successfully.

If some problems occur while making changes, you can restore your project from backup file. To do this, perform the following steps.

- Select **Migration Wizard** from the **ODAC** menu
- Select Restore original files from backup and press the Next button.
- Select the backup file. By default it is RExpert.reu file in RBackup folder of your converted project. Press the Next button.
- Check your settings and press the Finish button to start the conversion operation.
- Press **Yes** in the dialog that appeared.

Your project will be restored to its previous state.

Request Support DAC Forum Provide Feedback

3.3 Connecting to Oracle

Contents

Requirements

In order to connect to Oracle server you need the server itself running, ODAC installed and IDE running. If you have Oracle Client Software installed and want to use it, you need to know TNS alias name, login and password. If you do not wish to use OCI, you have to know host name or IP address, Oracle System Identifier (SID) or Oracle Servise Name, port, login and password.

General information

To establish a connection to server you have to provide some connection parameters to ODAC. This information is used by OraSession component to find the server and login with credentials of your account. The parameters are represented as connection string. You can compose the connection string manually or have ODAC construct it for you.

There are two ways to connect to server: with and without Oracle Client Interface. This is controlled by Direct property. It indicates whether the Oracle Client Interface will be used for connecting to server. By default Direct mode is disabled to preserve maximal functionality. Switch to Direct mode if you want to work in a system without Oracle Client Software installed.

Creating OraSession

Design time creation

The following assumes that you have IDE running, and you are currently focused on a form designer.

1. Find the OraSession component on the ODAC tab of the component palette.
2. Double-click the component. Notice that new object appears on the designer underneath the form. If this is first time you create the OraSession in this application it is named OraSession1.
3. Click on the OraSession1 object and press F11 to focus on object's properties. Or double-click on OraSession1 to open the dialog.
4. If you connect through OCI, in the Server property provide TNS alias of the server.
5. If you use Direct mode, perform the following assignments:
   - set Direct to true

- Set the Server property to a string that contains the host address of the database server, port number, and the Oracle System Identifier (SID) or Oracle Servise Name in the following format: Host:Port:SID or Host:Port:sn=ServiceName

6. In the Username property specify your login. For example, scott.
7. In the Password property specify your password. For example, tiger.

Run time creation

Same operations performed in runtime look as follows (note that you have to add DB, DBAccess, Ora units to the uses clause):

[Delphi OCI]

```
uses DB, DBAccess, Ora;
...
var
  OraSession1: TOraSession;
begin
  OraSession1 := TOraSession.Create(nil);
  OraSession1.Server := 'ORASERVER';
  OraSession1.Username := 'SCOTT';
  OraSession1.Password := 'TIGER';
```

[Delphi Direct]

```
uses DB, DBAccess, Ora;
...
var
  OraSession1: TOraSession;
begin
  OraSession1 := TOraSession.Create(nil);
  OraSession1.Options.Direct := True;
  OraSession1.Server := 'LOCALHOST:1521:ORASERVER';
  OraSession1.Username := 'SCOTT';
  OraSession1.Password := 'TIGER';
```

[C++ Builder OCI]

```
#pragma link "DBAccess"
#pragma link "Ora"
...
  TOraSession *OraSession1 = new TOraSession(NULL);
  OraSession1->Server = "ORASERVER";
  OraSession1->Username = "SCOT";
  OraSession1->Password = "TIGER";
```

[C++ Builder Direct]

```
#pragma link "DBAccess"
#pragma link "Ora"
```

```
...
  TOraSession *OraSession1 = new TOraSession(NULL);
  OraSession1->Options->Direct = True;
  OraSession1->Server = "LOCALHOST:1521:ORASERVER";
  OraSession1->Username = "SCOT";
  OraSession1->Password = "TIGER";
```

You can do this all in single assignment. It actually does not matter whether connection string is assigned directly or composed with particular properties. After you assign a value to ConnectionString property all other properties are populated with parsed values. So you can choose what is more convenient for you.

[Delphi OCI]

```
  OraSession1.ConnectString := 'SCOTT/TIGER@ORASERVER';
```

[Delphi Direct]

```
  OraSession1.ConnectString := 'SCOTT/TIGER@LOCALHOST:1521:ORASERVER';
```

[C++ Builder OCI]

```
  OraSession1->ConnectString = "SCOTT/TIGER@ORASERVER';
```

[C++ Builder Direct]

```
  OraSession1->ConnectString = "SCOTT/TIGER@LOCALHOST:1521:ORASERVER';
```

Opening connection

Opening a connection is as simple as that:

[Delphi]

```
  OraSession1.Connect;
```

[C++ Builder]

```
  OraSession1->Connect();
```

Of course, the OraSession1 must have valid connection string assigned earlier. When you call Connect, ODAC tries to find the host and connect to server. If any problem occurs it raises an exception with brief explanation on what is wrong. Finally, when connection is established, the Connect method returns and Connected property is changed to True.

In design time you can connect to server in few steps:

1. In the dialog window provide necessary logon information.
2. Click Connect button to establish connection.

Or you can simply change Connected property to True in Properties window to establish connection using current connection string.

Closing connection

To close a connection call its Disconnect method, or set its Connected property to False.

The following example summarizes aforementioned information and shows how to create, setup, open, use and then close the connection.

[Delphi]

```
var
  OraSession1: TOraSession;
begin
  OraSession1 := TOraSession.Create(nil);
  OraSession1.ConnectString := 'SCOTT/TIGER@ORASERVER';
  OraSession1.Connect;
  ShowMessage(OraSession1.OracleVersion);
  OraSession1.Disconnect;
```

[C++ Builder]

```
#pragma link "DBAccess"
#pragma link "Ora"
...
  TOraSession *OraSession1 = new TOraSession(NULL);
  OraSession1->Options->Direct = True;
  OraSession1->Server = "LOCALHOST:1521:ORASERVER";
  OraSession1->Username = "SCOT";
  OraSession1->Password = "TIGER";
```

Modifying connection

You can modify connection by changing properties of OraSession object. Keep in mind that while some of the properties can be altered freely, most of them close connection when new value is assigned. For example, if you change Server property, it gets closed immediately, and you have to reopen it manually.

Additional information

ODAC has wide set of features you can take advantage of. The following list enumerates some of them so you can explore the advanced techniques to achieve better performance, balance network load or enable additional capabilities.

Asynchronous connection opening Connection pooling (refer to MSDN documentation for

information about connection pooling).

- Asynchronous connection opening
- Connection pooling

See Also

- TOraSession

Request Support DAC Forum Provide Feedback

3.4 Creating Database Objects

This tutorial describes how to create tables, stored procedures and other objects in Oracle.

Contents

Requirements

In order to create database objects you have to connect to server. This process is described in details in Connecting to Oracle.

General information

Database objects are created using Data Definition Language (DDL), which is a part of SQL. The DDL statements can be executed on server by account that has necessary privileges.

There are two ways to manipulate a database. You can build DDL statements manually and run them within Oracle SQL*Plus or component like OraQuery. Another way is to use IDE - visual shells that provide graphical user interface to manage database. We will discuss both ways.

Using SQL*Plus

1. Launch the SQL*Plus and authorize yourself.
2. Type the following code and press Enter. This will create first of the tables we'll use for tutorial purposes. :

```
CREATE TABLE dept (
  deptno INT PRIMARY KEY,
  dname VARCHAR2(14),
  loc VARCHAR2(13)
)
```

3. Run the following query. This is another table we'll use.

```
CREATE TABLE emp (
  empno INT PRIMARY KEY,
  ename VARCHAR2(10),
  job VARCHAR2(9),
  mgr INT,
  hiredate DATE,
  sal FLOAT,
  comm FLOAT,
  deptno INT REFERENCES dept
)
```

4. These two tables are enough to demonstrate basic functionality. Now you can type exit to exit the SQL*Plus.

Additional information

Actually there are lots of ways to create tables on server. Any tool or component that is capable of running a SQL query, can be used to manage database objects. For example, OracleCommand suits fine for creating objects one by one, while OracleScript is designed for executing series of DDL/DML statements. For information on DDL statements syntax refer to Oracle documentation.

Request Support DAC Forum Provide Feedback

3.5 Retrieving and Modifying Data

Introducing

This tutorial describes how to use OraQuery component.

Requirements

This walkthrough supposes that you know how to connect to server, how to create the

necessary objects on the server, and how to insert the data to the created tables.

Retrieving and Updating Data

In this sample we are using OraQuery to retrieve and manipulate data. For more information, refer to the description of this class in our documentation.

[Delphi]

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  DB,
  DBAccess,
  Ora;
procedure PrintDept(OraSession: TOraSession);
var
  OraQuery:TOraQuery;
  i: integer;
begin
  OraQuery := TOraQuery.Create(nil);
  OraQuery.Session := OraSession;
  OraQuery.SQL.Text := 'SELECT * FROM dept';
  try
    OraQuery.Open;
    for i:= 0 to OraQuery.Fields.Count - 1 do
      Write(OraQuery.Fields[i].DisplayName+#09);
    WriteLn;
    while not OraQuery.Eof do
    begin
      for i:= 0 to OraQuery.Fields.Count - 1 do
        Write(OraQuery.Fields[i].AsString+#09);
      WriteLn;
      OraQuery.Next;
    end;
  finally
    OraQuery.close;
    OraQuery.Free;
  end;
end;
procedure ModifyDept(OraSession: TOraSession; SQLText: string; action: strin
var
  OraQuery:TOraQuery;
begin
  OraQuery := TOraQuery.Create(nil);
  OraQuery.Session := OraSession;
  OraQuery.SQL.Text := SQLText;
  try
    OraQuery.Execute;
    Writeln;
    write(format('Rows in DEPT %s: %d', [action,OraQuery.RowsAffected]));
  finally
    OraQuery.Free;
```

```
  end;
end;
var
  OraSession: TOraSession;
begin
  OraSession := TOraSession.Create(nil);
  OraSession.ConnectString := 'SCOTT/TIGER@ORCL1020';
  try
    OraSession.Connect;
    PrintDept(OraSession);
    ModifyDept(OraSession,'UPDATE DEPT SET LOC=''VEGAS'' WHERE DEPTNO > 20',
    ModifyDept(OraSession, 'INSERT INTO dept (deptno, dname, loc) VALUES (50
    ModifyDept(OraSession,'DELETE FROM dept WHERE deptno = 50','deleted');
    Readln;
  finally
    OraSession.Free;
  end;
end.
```

[C++ Builder]

```
#include <vcl.h>
#pragma hdrstop
#include <tchar.h>
#include <stdio.h>
#include <DBAccess.hpp>
#include <Ora.hpp>
#pragma argsused
void PrintDept(TOraSession *OraSession)
{
  TOraQuery *OraQuery = new TOraQuery(NULL);
  OraQuery->SQL->Text = "SELECT * FROM dept";
  OraQuery->Session = OraSession;
  int i;
  try
  {
 OraQuery->Open();
 for(i = 0; i< OraQuery->Fields->Count; i++)
   printf("%s\t",OraQuery->Fields->operator [](i)->DisplayName.t_str());
 printf("\n");
 while (!OraQuery->Eof)
 {
  for(i = 0; i< OraQuery->Fields->Count; i++)
   printf("%s\t",OraQuery->Fields->operator [](i)->AsString.t_str());
  printf("\n");
  OraQuery->Next();
 }
  }
  __finally
  {
 OraQuery->Free();
  }
}
void ModifyDept(TOraSession *OraSession, String SQLText, String Action)
{
  TOraQuery *OraQuery = new TOraQuery(NULL);
  OraQuery->SQL->Text = SQLText;
```

```
  OraQuery->Session = OraSession;
  try
  {
   OraQuery->Execute();
   printf("Rows in DEPT %s: %d", Action.t_str(), OraQuery->RowsAffected);
   printf("\n");
  }
  __finally
  {
 OraQuery->Free();
  }
}
int _tmain(int argc, _TCHAR* argv[])
{
 TOraSession *OraSession = new TOraSession(NULL);
 int i;
 try
 {
  OraSession->ConnectString = "SCOTT/TIGER@ORCL1020";
  OraSession->Connect();
  PrintDept(OraSession);
  ModifyDept(OraSession, "UPDATE DEPT SET LOC='VEGAS' WHERE DEPTNO > 20","up
  ModifyDept(OraSession, "INSERT INTO dept (deptno, dname, loc) VALUES (50,'
  ModifyDept(OraSession, "DELETE FROM dept WHERE deptno = 50","deleted");
  system("pause");
 }
 __finally
 {
  OraSession->Free();
 }
 return 0;
}
```


Request Support DAC Forum Provide Feedback

3.6 Inserting Data Into Tables

This tutorial describes how to use OraQuery component to insert data into tables by means of executing SQL queries.

- Requirements
- General information
- Inserting data in run time
- Design time setup
- Additional information

Requirements

This walkthrough supposes that you know how to connect to server (tutorial Logging onto the server) and that necessary objects are already created on the server (tutorial Creating database objects).

General information

Data on server can be modified (inserted, changed or deleted) using Data Manipulation Language (DML), which is a part of SQL. The DML statements can be executed on server by account that has necessary privileges.

There are two ways to manipulate a database. You can build DML statements manually and run them within some component like OraQuery. Another way is to use design-time features that provide graphical user interface to manage database. We will discuss both ways.

The goal of this tutorial is to insert the following data into tables dept and emp:
Table dept

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Table emp

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	17.12.1980	800	NULL	20
7499	ALLEN	SALESMAN	7698	20.02.1981	1600	300	30
7521	WAR	SALE	7698	22.02	1250	500	30

	D	SMAN		.1981			
7566	JONES	MANAGER	7839	02.04.1981	2975	NULL	20
7654	MARTIN	SALESMAN	7698	28.09.1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01.05.1981	2850	NULL	30
7782	CLARK	MANAGER	7839	09.06.1981	2450	NULL	10
7788	SCOTT	ANALYST	7566	13.07.1987	3000	NULL	20
7839	KING	PRESIDENT	NULL	17.11.1981	5000	NULL	10
7844	TURNER	SALESMAN	7698	08.09.1981	1500	0	30
7876	ADAMS	CLERK	7788	13.07.1987	1100	NULL	20
7900	JAMES	CLERK	7698	03.12.1981	950	NULL	30
7902	FORD	ANALYST	7566	03.12.1981	3000	NULL	20
7934	MILLER	CLERK	7782	23.01.1982	1300	NULL	10

Inserting data in run time

To insert the first row into table dept you can use the following statement:

INSERT INTO dept (deptno, dname, loc) VALUES (10,'ACCOUNTING','NEW YORK')

The following code fragment executes the query:

[Delphi]

```
var
  OraSession1: TOraSession;
  OraQuery1: TOraQuery;
begin
```

```
  OraSession1 := TOraSession.Create(nil);
  OraQuery1:= TOraQuery.Create(nil);
  OraSession1.ConnectString := 'SCOTT/TIGER@ORASERVER';
  OraQuery1.SQL.Text := 'INSERT INTO dept (deptno, dname, loc) VALUES (10,''
  OraQuery1.Session := OraSession1;
  OraSession1.LoginPrompt := False;
  try
    OraSession1.Connect;
    try
      OraQuery1.Execute;
      ShowMessage(IntToStr(OraQuery1.RowsAffected)+' rows were affected.');
    except
      ShowMessage('Error encountered during INSERT operation.');
    end;
  finally
    OraSession1.disconnect;
    OraQuery1.Free;
    OraSession1.Free;
  end;
```

[C++ Builder]

```
  TOraSession *OraSession1 = new TOraSession(NULL);
  TOraQuery *OraQuery1 = new TOraQuery(NULL);
  OraSession1->ConnectString = "SCOTT/TIGER@ORCL1020";
  OraQuery1->SQL->Text = "INSERT INTO dept (deptno, dname, loc) VALUES (10,'
  OraQuery1->Session = OraSession1;
  OraSession1->LoginPrompt = false;
  try
  {
OraSession1->Connect();
try
{
  OraQuery1->Execute();
  ShowMessage(IntToStr(OraQuery1->RowsAffected)+" rows were affected.");
}
catch(const Exception& e)
{
  ShowMessage("Error encountered during INSERT operation.");
}
 }
 __finally
 {
OraSession1->Disconnect();
OraQuery1->Free();
OraSession1->Free();
 }
```

The sample first creates a connection with hardcoded connection string. Then it creates OraQuery object, assigns the query text and connection to the OraQuery instance. Connection is opened then. The Execute method of OraQuery runs SQL statement in the Text property. The RowsAffected property stores the number of rows affected by the query. This method is not intended to run SELECT statements. We will discuss retrieving data in

other tutorials.

If the query is executed successfully you are notified about number of affected rows. If some error occurs you get the error message. The connection is closed anyway. It is recommended that you use `try ... finally` clauses to make sure the connections are closed properly.

Design time setup

Same operations in design time include following steps:

Place OraSession component on a designer.

1. Setup its properties and open connection by changing the Connected property to True or double-click on the component, in the dialog window provide necessary logon information and press Connect button.
2. Place OraQuery component on the designer.
3. In its Session property select name of the OraSession instance on the designer.
4. Click on the ellipsis in SQL property in Properties window double-click the component in the OraQuery editor on the SQL tab and enter the following query:

```
INSERT INTO dept VALUES (20,'SALES','DALLAS')
```

and press the Execute button.

Additional information

Actually there are lots of ways to insert data into tables. Any tool or component that is capable of running a SQL query, can be used to manage data. Some components are best for performing certain tasks. For example, OraLoader is the fastest way to insert data, OraScript is designed for executing series of statements. For more information on these components refer to ODAC reference.

See Also

Getting Started
OraQuery Class
OraLoader Class
OraScript Class

3.7 Working With Oracle Stored Procedures

This section describes how to create and use Oracle stored procedures and functions with ODAC.

It supposes that you know how to connect to server, how to create the necessary objects on the server, and how to manipulate with the data stored in database tables. The section describes stored procedures using for typed DataSets.

This section contains the following articles:

- Stored Procedures - General Information
- Contains general information about stored procedures and functions and describes how to create them.

- Using Stored Procedures via the TOraStoredProc Class
- Contains an information about using stored procedures with the help of TOraStoredProc class.

- Using Package Procedures
- Describes approaches of the stored procedure usage when they are included into the Oracle packages.

See Also

- Stored Procedures - General Information
- Using Stored Procedures via the TOraStoredProc class
- Using Package Procedures

3.7.1 Stored Procedures - General Information

This section contains information about general aspects of stored procedures usage.

A stored procedure is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or perform a set of related tasks. Procedures let you combine the ease and flexibility of SQL with the procedural functionality of a structured programming language. Large or complex processing that might require the execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

Objects similar to stored procedures are stored functions. Almost everything that is true for procedures, holds for functions as well. The main difference between these objects is that function has a return value, and procedure has not.

A stored procedures and functions may have input, output, and input/output parameters.

Input parameter is a parameter whose value is passed into a stored procedure/function module. The value of an IN parameter is a constant; it can't be changed or reassigned within the module.

For example, the following procedure inserts a row into the Dept table:

```
CREATE PROCEDURE dept_insert (pDeptno INTEGER, pDname VARCHAR2, pLoc VARCHAR
BEGIN
  INSERT INTO dept(deptno, dname, loc) VALUES (pDeptno, pDname, pLoc);
END;
```

It needs to receive the values to be inserted into the new record, and thus the procedure has three input parameters, corresponding to each field of the table. The procedure may be executed inside a PL/SQL block like follows:

```
begin
  dept_insert (10, 'Accounting', 'New York');
end;
```

Output parameter is a parameter whose value is passed out of the stored procedure/function module, back to the calling PL/SQL block. An OUT parameter must be a variable, not a constant. It can be found only on the left-hand side of an assignment in the module. You cannot assign a default value to an OUT parameter outside of the module's body. In other words, an OUT parameter behaves like an uninitialized variable. In the following sample, the stored procedure returns the count of records in table Dept:

```
CREATE PROCEDURE dept_count (cnt OUT INTEGER)
AS
```

```
BEGIN
  SELECT COUNT(*) INTO cnt FROM dept;
END;
```

An input/output parameter is a parameter that functions as an IN or an OUT parameter or both. The value of the IN/OUT parameter is passed into the stored procedure/function and a new value can be assigned to the parameter and passed out of the module. An IN/OUT parameter must be a variable, not a constant. However, it can be found on both sides of an assignment. In other words, an IN/OUT parameter behaves like an initialized variable.

Besides scalar variables, a stored procedure can return result sets, i.e. the results of a SELECT statement. In Oracle, the cursor variables are used for this case. A cursor may be interpreted as a reference to the result set. The following sample demonstrates how a simplest select statement can be wrapped in a stored procedure:

```
CREATE PROCEDURE get_all_depts_proc (cur OUT SYS_REFCURSOR) AS
BEGIN
  OPEN cur FOR SELECT * FROM dept;
END;
```

The same SELECT statement can be used via a stored function as follows:

```
CREATE OR REPLACE FUNCTION get_all_depts_func RETURN SYS_REFCURSOR
AS
  cur SYS_REFCURSOR;
BEGIN
  OPEN cur FOR SELECT * FROM dept;
  RETURN cur;
END;
```

Here the cursor is passed as the return value instead of being an output parameter.

See Also:

- Using Stored Procedures via the TOraStoredProc class
- Package Procedures

Request Support DAC Forum Provide Feedback

3.7.2 Using Stored Procedures via the TOraStoredProc Class

This topic describes how to use Oracle stored procedures and functions with ODAC by the help of TOraStoredProc class.

The following sample demonstrates the work with an Oracle stored procedure using the

get_all_depts_proc procedure from the previous section. Since the out parameter of the procedure is the cursor, it is possible to work with the procedure as with a simple DataSet.

Note: If several out parameters in the procedure are cursors, then TOraStoredProc will work only with the first one of them as with a DataSet.

[Delphi]

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  DB,
  DBAccess,
  Ora;
procedure PrintDept(OraSession: TOraSession);
var
  OraStoredProc: TOraStoredProc;
  i: integer;
begin
  //procedure creation
  OraStoredProc := TOraStoredProc.Create(nil);
  OraStoredProc.Session := OraSession;
  //setting the stored procedure name
  OraStoredProc.StoredProcName := 'get_all_depts_proc';
  //The ParamCheck property must be set to True for automatic
  //definition of parameters used in the stored procedure
  OraStoredProc.ParamCheck := True;
  try
    //execution of the stored procedure
    OraStoredProc.Execute;
    //retrieving data from the cursor returned by the procedure
    for i:= 0 to OraStoredProc.FieldCount - 1 do
      Write(OraStoredProc.Fields[i].DisplayName+#09);
    WriteLn;
    while not OraStoredProc.Eof do
    begin
      for i:= 0 to OraStoredProc.Fields.Count - 1 do
        Write(OraStoredProc.Fields[i].AsString+#09);
      WriteLn;
      OraStoredProc.Next;
    end;
  finally
    OraStoredProc.close;
    OraStoredProc.Free;
  end;
end;
var
  OraSession: TOraSession;
begin
  OraSession := TOraSession.Create(nil);
  OraSession.ConnectString := 'SCOTT/TIGER@ORCL1020';
  try
```

```
    OraSession.Connect;
    PrintDept(OraSession);
    readln;
  finally
    OraSession.Free;
  end;
end.
```

[C++ Builder]

```
#include <vcl.h>
#pragma hdrstop
#include <tchar.h>
#include <stdio.h>
#include <DBAccess.hpp>
#include <Ora.hpp>
#pragma argsused
void PrintDept(TOraSession *OraSession)
{
  TOraStoredProc *OraStoredProc = new TOraStoredProc(NULL);
  OraStoredProc->StoredProcName = "get_all_depts_proc";
  OraStoredProc->Session = OraSession;
  int i;
  try
  {
 OraStoredProc->Execute();
 for(i = 0; i< OraStoredProc->Fields->Count; i++)
   printf("%s\t",OraStoredProc->Fields->operator [](i)->DisplayName.t_str())
 printf("\n");
 while (!OraStoredProc->Eof)
 {
  for(i = 0; i< OraStoredProc->Fields->Count; i++)
   printf("%s\t",OraStoredProc->Fields->operator [](i)->AsString.t_str());
  printf("\n");
  OraStoredProc->Next();
 }
  }
  __finally
  {
 OraStoredProc->Free();
  }
}
int _tmain(int argc, _TCHAR* argv[])
{
 TOraSession *OraSession = new TOraSession(NULL);
 int i;
 try
 {
  OraSession->ConnectString = "SCOTT/TIGER@ORCL1020";
  OraSession->Connect();
  PrintDept(OraSession);
  system("pause");
 }
 __finally
 {
  OraSession->Free();
 }
```

```
 return 0;
}
```

See Also

- Stored Procedures - General Information
- Using Package Procedures

 Request Support DAC Forum Provide Feedback

3.7.3 Using Package Procedures

This topic describes how to create and use Oracle stored procedures and functions within the Oracle packages with ODAC.

In Oracle databases, stored procedures and functions may be grouped into specific sets which are called packages. To call a package procedure, one needs only to add the package name before the procedure's name, like "package.get_all_depts_proc". However, with ODAC using of packages may be even easier due to typed and untyped. The first ones are the classes generated by the Package Wizard, the second ones are instances of the TOraPackage class.

Untyped OraPackage may be set to represent any package specified by the name, provided that this package is available for the connection used. TOraPackage class has a set of methods intended to execute procedures and retrieve their descriptions.

Typed OraPackage is a class representing the only specific package. For each procedure or function of this package, instances of corresponding typed OraPackage have a special method. Such approach allows to invoke stored procedures just like usual object methods. Typed Oracle packages can be created using Oracle Package Wizard. For more information on this see Using Package Wizard for working with PL/SQL Packages.

See Also

- Stored Procedures - General Information
- Using Stored Procedures via the TOraStoredProc class

 Request Support DAC Forum Provide Feedback

3.8 Working With PL/SQL

This section describes how to create and use Oracle PL/SQL blocks ODAC.

It supposes that you know how to connect to server, how to create the necessary objects on the server, and how to manipulate with the data stored in database tables. The section describes stored procedures using for typed DataSets.

This section contains the following articles:

- PL/SQL - General Information

Contains general information PL/SQL blocks and describes how to create them.

- Using PL/SQL via the TOraSQL Class

Contains an information about using PL/SQL with the help of TOraSQL class.

 Request Support DAC Forum Provide Feedback

3.8.1 PL/SQL - General Information

PL/SQL (Procedural Language/Structured Query Language, also known as Pretty Lazy/ Structured Query Language) is Oracle Corporation's procedural extension language for SQL and the Oracle relational database. PL/SQL's general syntax resembles that of Ada. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java. PL/SQL is available in Oracle Database (since version 7).

- Introduction
- Basic Code Structure
- Data Types
- Control Operators
- Application Sample

Introduction

PL/SQL supports variables, conditions, loops and exceptions. Arrays are also supported, though in a somewhat unusual way, involving the use of PL/SQL collections. PL/SQL

collections are a slightly advanced topic. Implementations from version 8 of Oracle Database onwards have included features associated with object-orientation. PL/SQL program units (essentially code containers) can be compiled into the Oracle database. Programmers can thus embed PL/SQL units of functionality into the database directly. They also can write scripts containing PL/SQL program units that can be read into the database using the Oracle SQL*Plus tool. Once the program units have been stored into the database, they become available for execution at a later time. While programmers can readily embed Data Manipulation Language (DML) statements directly into their PL/SQL code using straight forward SQL statements, Data Definition Language (DDL) requires more complex "Dynamic SQL" statements to be written in the PL/SQL code. However, DML statements underpin the majority of PL/SQL code in typical software applications. In the case of PL/SQL dynamic SQL, early versions of the Oracle Database required the use of a complicated Oracle DBMS_SQL package library. More recent versions have however introduced a simpler "Native Dynamic SQL", along with an associated EXECUTE IMMEDIATE syntax. Oracle Corporation customarily extends package functionality with each successive release of the Oracle Database.

Basic Code Structure

An application on PL/SQL consists of blocks (anonymous and named). A block can include nested blocks, aka subblocks. The general shape of a PL/SQL-block:

```
<<label>>
DECLARE
        TYPE / item / FUNCTION / PROCEDURE declarations
BEGIN
      Statements
EXCEPTION
        EXCEPTION handlers
END label;
```

Data Types

The PL/SQL language supports the following type categories:

- nested data types, including collections and records;
- scalar;
- compound;
- reference;

- LOB-types;
- Object data types.

Control Operators

- selection statements:

```
IF - THEN - END IF;
IF - THEN - ELSE - END IF;
IF - THEN - ELSIF - END IF;
CASE - WHEN - THEN - END CASE;
```

- loop statements:

```
LOOP - END LOOP;
WHILE - LOOP - END LOOP;
FOR - LOOP - END LOOP;
EXIT;
EXIT WHEN;
```

- GO TO statements:

```
GOTO label_name;
```

Application Sample

Example of a program that updates data in the table and displays the number of changed records in the console

```
DECLARE
 cnt NUMBER;
BEGIN
 SELECT DEPTNO
 INTO cnt
  FROM DEPT
  WHERE DEPTNO = 10;
 UPDATE DEPT SET LOC='VEGAS' WHERE DEPTNO = 10;
 DBMS_OUTPUT.PUT_LINE('Rows in DEPT updated '||sql%rowcount);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Row for update in DEPT no found') ;
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.put_line('Query returm more that one row');
END;
```

See also:

- Using PL/SQL via the TOraSQL Class

Request Support DAC Forum Provide Feedback

3.8.2 Using PL/SQL via the TOraSQL Class

This topic describes how to use PL/SQL with ODAC by the help of TOraSQL class.

For work with PL/SQL blocks, the TOraSQL component is used, that allows using parameters and macros in PL/SQL code.

A sample application updating data in a table and returning the number of modified records to the console:

[Delphi]

```
  OraSQL1.SQL.Text := 'DECLARE' + #13#10 +
                      '  cnt NUMBER;' + #13#10 +
                       'BEGIN' + #13#10 +
                      'SELECT DEPTNO' + #13#10 +
                       '  INTO cnt' + #13#10 +
                     'FROM DEPT' + #13#10 +
                     'WHERE DNAME = :DNAME;' + #13#10 +
                       '  UPDATE EMP SET SAL=SAL + 100 WHERE DEPTNO = cnt;' +
                       '  :RES := sql%rowcount;' + #13#10 +
                       'EXCEPTION' + #13#10 +
                       '  WHEN NO_DATA_FOUND THEN' + #13#10 +
                    '    DBMS_OUTPUT.PUT_LINE(''Row for update in EMP no foun
                       '  WHEN TOO_MANY_ROWS THEN' + #13#10 +
                       '    DBMS_OUTPUT.put_line(''Query returm more that one
                       'END;';
  OraSQL1.ParamByName('DNAME').DataType := ftString;
  OraSQL1.ParamByName('DNAME').ParamType := ptInput;
  OraSQL1.ParamByName('DNAME').AsString := 'ACCOUNT';
  OraSQL1.ParamByName('RES').DataType := ftInteger;
  OraSQL1.ParamByName('RES').ParamType := ptOutput;
  OraSQL1.Execute;
  ShowMessage(Format('Rows in EMP updated: %d',[OraSQL1.ParamByName('RES').A
```

[C++ Builder]

```
  OraSQL1->SQL->Text = "DECLARE\n"
      "  cnt NUMBER;\n"
      "BEGIN\n"
      "SELECT DEPTNO\n"
      "  INTO cnt\n"
      "FROM DEPT\n"
      "WHERE DNAME = :DNAME;\n"
```

```
      "  UPDATE EMP SET SAL=SAL + 100 WHERE DEPTNO = cnt;\n"
      "  :RES := sql%rowcount;\n"
      "EXCEPTION\n"
      "  WHEN NO_DATA_FOUND THEN\n"
      "    DBMS_OUTPUT->PUT_LINE('Row for update in EMP no found');\n"
      "  WHEN TOO_MANY_ROWS THEN\n"
      "    DBMS_OUTPUT->put_line('Query returm more that one row');\n"
      "END;";
  OraSQL1->ParamByName("DNAME")->DataType = ftString;
  OraSQL1->ParamByName("DNAME")->ParamType = ptInput;
  OraSQL1->ParamByName("DNAME")->AsString = "ACCOUNT";
  OraSQL1->ParamByName("RES")->DataType = ftInteger;
  OraSQL1->ParamByName("RES")->ParamType = ptOutput;
  OraSQL1->Execute();
```


Request Support DAC Forum Provide Feedback

3.9 Using Transactions

Understanding Transactions

A transaction is one or several operations considered as a single unit of work which is completed entirely or have no effect at all ("all-or-nothing"). If a failure occurs at one point in the transaction, all of the updates can be rolled back to their pre-transaction state. A transaction must conform to the ACID properties - atomicity, consistency, isolation, and durability-in order to guarantee data consistency.

If a transaction involves multiple tables in the same database, then explicit transactions in PL/SQL often perform better. You can use COMMIT and ROLLBACK statements in your SQL to fix and discard respectively the previous commands in your current PL/SQL block. For more information, see Oracle PL/SQL documentation.

Otherwise, a transaction with plain SQL can be implemented via TOraTransaction TOraSession classes. For example, you can use TOraSession: start transaction on TOraSession, execute several SQL statements via TOraDataSet, and commit/rollback all operations when it is necessary. See the sample from the Local Transaction topic.

This article describes the way to manipulate transactions from your application (without envolving PL/SQL transactions) - this is the most common case of working with transactions. Concerning your task, you can choose the type of transaction to implement - local or distributed. A transaction considered to be a local transaction when it is a single-phase transaction and is handled by the database directly. A distributed transaction is a transaction that affects several resources, it is coordinated by a transaction monitor and uses fail-safe

mechanisms (such as two-phase commit) for transaction resolution.

Note: transaction will be global if either TransactionId or TransactionName property is set or if GlobalCoordinator property is gcMTS.

Local Transactions

To start local transaction with TOraTransaction component, set DefaultSession property of the component to a session on which transaction will be performed. Set IsolationLevel property optionally. Then call StartTransaction method of the TOraTransaction component. To manage transaction use Commit, Rollback, Savepoint, RollbackToSavepoint methods.

[Delphi]

```
var
  OraSession: TOraSession;
  OraTransaction: TOraTransaction;
begin
  OraSession := TOraSession.Create(nil);
  OraTransaction := TOraTransaction.Create(nil);
  try
    OraSession.ConnectString := 'login/password@SID';
    OraSession.Connect;
    OraTransaction.AddSession(OraSession);
    OraTransaction.IsolationLevel := ilReadCommitted;
    OraTransaction.StartTransaction;
    try
      OraSession.ExecSQL('INSERT INTO Dept(DeptNo, DName) Values(50, ''DEVEL
      OraSession.ExecSQL('INSERT INTO Dept(DeptNo, DName) Values(60, ''PRODU
      OraTransaction.Commit;
      ShowMessage('Both records are written to database.');
    except
      OraTransaction.Rollback;
      ShowMessage('Neither record was written to database.');
    end;
  finally
    OraTransaction.Free;
    OraSession.Free;
  end;
end;
```

[C++ Builder]

```
  TOraSession *OraSession = new TOraSession(NULL);
  TOraTransaction *OraTransaction = new TOraTransaction(NULL);
  try
  {
 OraSession->ConnectString = "SCOTT/TIGER@ORCL1020";
 OraSession->Connect();
 OraTransaction->AddSession(OraSession);
 OraTransaction->IsolationLevel = ilReadCommitted;
 OraTransaction->StartTransaction();
```

```
try
{
  OraSession->ExecSQL("INSERT INTO Dept(DeptNo, DName) Values(50, 'DEVELOPM
  OraSession->ExecSQL("INSERT INTO Dept(DeptNo, DName) Values(60, 'PRODUCTI
  ShowMessage("Both records are written to database.");
}
catch(const Exception& e)
{
  OraTransaction->Rollback();
  ShowMessage("Neither record was written to database.");
}
 }
 __finally
 {
OraSession->Disconnect();
OraTransaction->Free();
OraSession->Free();
 }
```


Request Support DAC Forum Provide Feedback

3.10 Demo Projects

ODAC includes a number of demo projects that show off the main ODAC functionality and development patterns.

The ODAC demo projects consist of one large project called *OdacDemo* with demos for all main ODAC components, use cases, and data access technologies, and a number of smaller projects on how to use ODAC in different IDEs and how to integrate ODAC with third-party components.

Most demo projects are built for Delphi and Embarcadero RAD Studio. There are only two ODAC demos for C++Builder. However, the C++Builder distribution includes source code for all other demo projects as well.

Where are the ODAC demo projects located?

In most cases all the ODAC demo projects are located in "%Odac%\Demos\".

In Delphi 2007 for Win32 under Windows Vista all the ODAC demo projects are located in "My Documents\Devart\Odac for Delphi 2007\Demos", for example "C:\Documents and Settings\All Users\Documents\Devart\Odac for Delphi 2007\Demos\".

The structure of the demo project directory depends on the IDE version you are using.

For most new IDEs the structure will be as following.

```
Demos
  |—OdacDemo [The main ODAC demo project]
  |—Performance [Demo project, that compares performance of ODAC
  with another components (BDE, ADO, dbExpress)]
  |—ThirdParty
  |   |— [A collection of demo projects on integration with third-
  party components]
  |—Miscellaneous
     |— [Some other demo projects on design technologies]
```

OdacDemo is the main demo project that shows off all the ODAC functionality. The other directories contain a number of supplementary demo projects that describe special use cases. The list of all samples in the ODAC demo project and the description for the supplementary projects is provided in the following section.

Note: This documentation describes ALL the ODAC demo projects. The actual demo projects you will have installed on your computer depends on your ODAC version, ODAC edition, and the IDE version you are using. The integration demos may require installation of third-party components to compile and work properly.

Instructions for using the ODAC demo projects

To explore an ODAC demo project,

1. Launch your IDE.
2. In your IDE, choose File|Open Project from the menu bar.
3. Find the directory you have installed ODAC to and open the Demos folder.
4. Browse through the demo project folders located here and open the project file of the demo you would like to use.
5. Compile and launch the demo. If it exists, consult the *ReadMe.txt* file for more details.

The executed version of the demo will contain a sample application written with ODAC or a navigable list of samples and sample descriptions. To use each sample properly, you will need to connect to a working Oracle server.

The included sample applications are fully functional. To use the demos, you have to set up a connection to Oracle first. You can do that by clicking on the "Connect" button.

Many demos may also use some database objects. If so, they will have two object manipulation buttons, "Create" and "Drop". If your demo requires additional objects, click "Create" to create necessary database objects. When you are done with the demo, click "Drop" to remove all the objects used for the demo from your Oracle database.

Note: The ODAC demo directory includes two sample SQL scripts for creating and dropping all the test schema objects used in the ODAC demos. You can modify and execute this script manually, if you want. This will not change the behavior of the demos.

You can find a complete walkthrough for the main ODAC demo project in the Getting Started topic. Other ODAC demo projects include a *ReadMe.txt* file with individual building and launching instructions.

Demo project descriptions

OdacDemo

*OdacDemo*is one large project which includes three collections of demos.

Working with components

A collection of samples that show how to work with the basic ODAC components.

General demos

A collection of samples that show off the ODAC technology and demonstrate some ways to work with data.

Oracle-specific demos

A collection of samples that demonstrate how to incorporate Oracle features in database applications.

OdacDemo can be opened from %Odac%\Demos\OdacDemo\odacdemo.dpr (.bdsproj). The following table describes all demos contained in this project.

Working with Components

Name	Description
Alerter	Uses the TOraAlerter component to send messages between sessions through DBMS_ALERT and DBMS_PIPE Oracle package functionality. Note: Requires execute privileges on DBMS_ALERT.
ChangeNotification	Demonstrates how to subscribe, receive, and reflect DML or DDL changes on objects associated with queries. Note: Requires CHANGE NOTIFICATION privilege. This functionallity is available only for Oracle 10.2g or higher.
ConnectDialog	Demonstrates how to customize the ODAC connect dialog . Changes the standard ODAC connect dialog to two custom connect dialogs. The first customized sample dialog is inherited from the TForm class, and the second one is inherited from the default ODAC connect dialog class.
CRDBGrid	Demonstrates how to work with the TCRDBGrid component. Shows off main TCRDBGrid features, like filtering, searching, stretching, using compound headers, and more.
ErrorHandler	Demonstrates using the TOraErrorHandler for exception handling and translating error messages.
Loader	Uses the TOraLoader component to load data into a server table quickly. Demonstrates Direct and DML modes for loading data. In Direct mode, data is loaded through DirectPath API, which loads big volumes of data into a table faster than while using INSERT statement. In DML mode, data is processed with the DML array feature of Oracle. It also compares two TOraLoader data loading handlers: GetColumnData and PutData.
Query	Demonstrates working with TOraQuery , which is one of the most useful ODAC components. Includes many TOraQuery usage scenarios. Demonstrates how to execute queries in both standard and NonBlocking mode and how to edit data and export it to XML files. Note: This is a very good introductory demo. We recommend starting with it when first becoming familiar with ODAC.
Queue	Demonstrates how to use ODAC to work with Oracle Streams Advanced Queuing. Implements notification on new messages. Shows how to create and manage Oracle Queues. Demonstrates the TOraQueue , TOraQueueTable , and TOraQueueAdmin components. Note: Requires DBMS_AQ and DBMS_AQADM privileges.
Smart	Uses TSmartQuery to customize refreshing, sorting, and server-side record management in a data grid. Shows how to perform local filtering, demonstrates several different kinds of record locking and refreshing, and working with FetchAll mode.
Sql	Uses TOraSQL to execute SQL statements and PL/SQL blocks. Demonstrates how to work in standard and NonBlocking modes, how to work with parameters in SQL, and how to break long-duration query

	execution.
StoredProc	Uses TOraStoredProc to access an editable recordset represented by an Oracle cursor from an Oracle stored procedure in the client application.
Table	Demonstrates how to use TOraTable to work with data from a single table on the server without writing any SQL queries manually. Performs server-side data sorting and filtering and retrieves results for browsing and editing.
Trace	Uses TOraTrace to work with Oracle SQL and PL/SQL tracing.
Transaction	Demonstrates main approaches for setting up distributed transactions with the TOraTransaction component. Shows how to manage transactions, tune the transaction isolation level, and select the coordinator for a distributed transaction.
UpdateSQL	Demonstrates using the TOraUpdateSQL component to customize update commands. Lets you optionally use TOraSQL and TOraQuery objects for carrying out insert, delete, query, and update commands.
VirtualTable	Demonstrates working with the TVirtualTable component. This sample shows how to fill virtual dataset with data from other datasets, filter data by a given criteria, locate specified records, perform file operations, and change data and table structure.

General Demos

Name	Description
CachedUpdates	Demonstrates how to perform the most important tasks of working with data in CachedUpdates mode, including highlighting uncommitted changes, managing transactions, and committing changes in a batch.
FilterAndIndex	Demonstrates ODAC's local storage functionality. This sample shows how to perform local filtering, sorting and locating by multiple fields, including by calculated and lookup fields.
MasterDetail	Uses ODAC functionality to work with master/detail relationships. This sample shows how to use local master/detail functionality. Demonstrates different kinds of master/detail linking, including linking by SQL, simple fields, and calculated fields.
Pictures	Uses ODAC functionality to work with BLOB fields and graphics. The sample demonstrates how to retrieve binary data from Oracle database and display it on visual components. Sample also shows how to load and save pictures to files and to the database.
Threads	Demonstrates how ODAC can be used in multithreaded applications. This sample allows you to set up several threads and test ODAC's performance with multithreading.

Oracle-specific Demos

Name	Description
Arrays	Demonstrates working with Oracle arrays . This sample lets you view and control how arrays are represented in dataset fields by the SparseArrays and ObjectView properties.
BFile	Shows the basics of working with file binary data stored in file systems located outside Oracle databases. This sample uses the TBFileField field type of ODAC.
BlobPictures	Demonstrates working with Oracle BLOB data types. The sample shows how to get binary data from the table, how to change BLOB fields using UPDATE statements, and how to insert a new record by executing stored procedure with a BLOB parameter. Also it shows off some extended BLOB handling functionality like local caching control, compression type changing, and more.
Clob	Demonstrates working with Oracle CLOB data types. The sample shows how to get a character stream from a table, how to change CLOB fields using UPDATE statements, and how to save and load data to/from a file. It also demonstrates several different ways of performing CLOB insertion.
Cursor	Uses ODAC functionality to work with Oracle Cursors . Shows how to fetch data from a Cursor parameter by setting TOraDataSet.Cursor to the TOraParam.AsCursor .
DMLArray	Demonstrates how to multiply execute SQL statements with different parameters by using ODAC functionality for the Oracle DML array feature.
FetchCursors	Uses TOraQuery to retrieve several Oracle cursors at once, fetch data in a batch, and close the cursors.
Long	Demonstrates working with Oracle LONG data types. The sample shows how to get character string from a table, update LONG fields and insert a new record. Also shows how to perform file operations with LONG fields.
LongStrings	Demonstrates ODAC functionality for working with long string fields (fields that have more than 256 characters). Shows different ways of their displaying as memo fields and string fields.
MultiCursors	Shows how to use one TOraQuery object to retrieve and update data from several tables by using several REF CURSOR parameters.
MultiQueries	Shows how Oracle queries are handled in multithreaded applications. This sample project lets you compare opening multiple queries within a single session or different sessions by setting NonBlocking and FetchAll =False modes.
NestedTables	Demonstrates using the TOraNestedTable component to work with Oracle nested tables. This sample project fills a TOraNestedTable dataset instance with the result set of a query to a table with a nested

	table field. It shows how to work with the data contained in nested table fields. Note that the nested table accessing interface is similar to the interface for accessing cursor data in a dataset representation of a table with CURSOR fields.
Objects	Demonstrates working with Oracle object fields. This sample shows how to clone objects and access and modify object field properties.
Pipes	Uses TOraAlerter in Pipe mode to organize cross-session message exchanging.
PLSQLTable	Demonstrates using PL/SQL Table types as parameters. Working with PL/SQL Table types in PL/SQL lets you imitate array functionality provided by other programming languages.
Progressor	Uses TOraAlerter in Pipe mode to indicate the progress of long-duration Oracle processes.
ProxySession	Demonstrates connecting to Oracle with the Oracle Proxy session functionality. This type of connection allows to quickly establish connections without specifying a password. Note: Requires the CONNECT THROUGH privilege
Refs	Demonstrates using the REF Oracle data type in queries.
SmartRefresh	Uses the Smart Refresh features of TSmartQuery to notify your data changes to other subscribed users. This sample is based on the TOraAlerter component. Note: Smart Refresh is only available in ODAC Professional Edition and ODAC Developer Edition.
XMLType	Uses the TOraXMLField class to work with Oracle SYS.XMLTYPE type LOB or Schema-based data. This sample project shows how to perform all the basic operations with this data type.

Supplementary Demo Projects

ODAC also includes a number of additional demo projects that describe some special use cases, show how to use ODAC in different IDEs and give examples of integrating it with third-party components. These supplementary ODAC demo projects are sorted into subfolders in the %Odac%\Demos\ directory.

Location	Name	Description
ThirdParty	**FastReport**	Demonstrates how ODAC can be used with FastReport components. This project consists of two parts. The first part consists of several packages that integrate ODAC components into the FastReport editor. The second part is a demo application that lets you design and

		preview reports with ODAC technology in the FastReport editor.
	InfoPower	Uses InfoPower components to display recordsets retrieved with ODAC. This demo project displays an InfoPower grid component and fills it with the result of an ODAC query. Shows how to link ODAC data sources to InfoPower components.
	IntraWeb	A collection of sample projects that show how to use ODAC components as data sources for IntraWeb applications. Contains IntraWeb samples for setting up a connection, querying a database and modifying data and working with CachedUpdates and MasterDetail relationships. Starting with Oracle 10.2g and higher lets you see the effect of setting TOraCachedUpdates.
	QuickReport	Lets you launch and view a QuickReport application based on ODAC. This demo project lets you modify the application in design-time.
	ReportBuilder	Uses ODAC data sources to create a ReportBuilder report that takes data from an Oracle database. Shows how to set up a ReportBuilder document in design-time and how to integrate ODAC components into the Report Builder editor to perform document design in run-time.
Miscellaneous	**CBuilder**	General demo project that shows how to create ODAC-based applications with C++Builder. Lets you execute SQL scripts and work with result sets in a grid. This is one of the two ODAC demos for C++Builder.
	Dll	Demonstrates creating and loading DLLs for ODAC-based projects.

		This demo project consists of two parts - an OraDll project that creates a DLL of a form that sends a query to the server and displays its results, and an OraExe project that can be executed to display a form for loading and running this DLL. Allows you to build a dll for one ODAC-based project and load and test it from a separate application.
	ExternalProc	Uses external procedures to save LOB data to file on an Oracle server and store the file name and file date in a database. This demo project uses the external procedure DLL file described in the Writing Oracle external procedures with ODAC topic.
	FailOver	Demonstrates the recommended approach to working with unstable networks. This sample lets you perform transactions and updates in several different modes, simulate sudden session termination, and view what happens to your data state when connections to the server are unexpectedly lost. Shows off CachedUpdates, LocalMasterDetail, FetchAll, Pooling, and different Failover modes.
	Geometry	Demonstrates using the SDO_GEOMETRY Oracle type with ODAC. This project reads SDO_GEOMETRY objects from the database and draws figures stored in these objects. The project allows you to edit figures and save them to the database.
	Midas	Demonstrates using MIDAS technology with ODAC. This project consists of two parts: a MIDAS server that processes requests to the database and a thin MIDAS client that displays an interactive grid. This demo shows how to build

		thin clients that display interactive components and delegate all database interactions to the server application for processing.
	Performance	Measures ODAC performance on several types of queries. This project lets you compare ODAC performance to BDE, ADO, and dbExpress. Tests the following functionality: Fetch, Master/Detail, Stored Procedure Call, Data Loading, Multi Executing, and Insert/Post.
	VirtualTableCB	Demonstrates working with the TVirtualTable component. This sample shows how to fill virtual dataset with data from other datasets, filter data by a given criteria, locate specified records, perform file operations, and change data and table structure. This is one of the two demo projects for C++Builder.
OdacDemo	***OdacDemo***	*[Win32 version of the main ODAC demo project - see above]*

Request Support DAC Forum Provide Feedback

3.11 Deployment

ODAC applications can be built and deployed with or without run-time libraries. Using run-time libraries is managed with the "Build with runtime packages" check box in the Project Options dialog box.

Deploying Windows applications built without run-time packages

You do not need to deploy any files with ODAC-based applications built without run-time packages, provided you are using a registered version of ODAC.

You can check your application does not require run-time packages by making sure the "Build

with runtime packages" check box is not selected in the Project Options dialog box.

Trial Limitation Warning

If you are evaluating deploying Windows applications with ODAC Trial Edition, you will need to deploy the following DAC BPL files:

dacXX.bpl	always
odacXX.bpl	always

and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages:

rtlXX.bpl	always
dbrtlXX.bpl	always
vcldbXXX.bpl	always

Deploying Windows applications built with run-time packages

You can set your application to be built with run-time packages by selecting the "Build with runtime packages" check box in the Project Options dialog box before compiling your application.

In this case, you will also need to deploy the following BPL files with your Windows application:

dacXX.bpl	always
odacXX.bpl	always
dacvclXX.bpl	if your application uses the OdacVcl unit
odacvclXX.bpl	if your application uses the OdacVcl unit
crcontrolsXX.bpl	if your application uses the CRDBGrid component

Request Support DAC Forum Provide Feedback

4 Using ODAC

This section describes basics of using Oracle Data Access Components

- Updating Data with ODAC Dataset Components

- Connecting in Direct Mode
- Master/Detail Relationships
- Data Type Mapping
- Data Encryption
- Working in an Unstable Network
- Secure Connections
- Disconnected Mode
- Increasing Performance
- Macros
- DataSet Manager
- Connection Pooling
- Automatic Key Field Value Generation
- TOraLoader Component
- TOraTransaction Component
- TOraQueue, TOraQueueAdmin and TOraQueueTable Components
- TOraChangeNotification Component
- BLOB and CLOB Data Types
- Unicode Character Data
- Objects
- XMLTYPE Data Type
- VARRAY Data Type
- DML Array
- PL/SQL Tables
- Writing Oracle External Procedures with ODAC
- Transparent Application Failover Support
- DBMonitor
- Writing GUI Applications with ODAC

- Compatibility with Previous Versions
- Integration with dbForge Studio for Oracle
- 64-bit Development with Embarcadero RAD Studio XE2
- Database Specific Aspects of 64-bit Development
- Demo Projects
- Deployment

Request Support DAC Forum Provide Feedback

4.1 Updating Data with ODAC Dataset Components

ODAC dataset components which descend from TCustomDADataSet provide different ways for reflecting local changes on the server.

The first approach is to use automatic generation of update SQL statements. When using this approach you should either specify Key Fields (the KeyFields property) or include RowID field into you SELECT SQL statement to avoid requesting of KeyFields from the server. When SELECT statement uses multiple tables, you can use UpdatingTable property to specify which table will be updated. If UpdatingTable is blank, the first table of the FROM clause will be used. When using sophisticated SELECT SQL statements (statements that use multiple tables, Synonyms, DBLinks, aggregated fields) we recommend to enable ExtendedFieldsInfo option. When this option is enabled, additional requests to the server may be performed to obtain more information about updating objects. This helps to generate correct updating SQL statements but may result in performance decrease. To avoid editing the fields that will not be used in update SQL statements use the SetFieldsReadOnly option. You can increase performance by refreshing fields using RETURNING clause when insert or update is performed. To enable this feature enable DMLRefresh and ReturnParams options.

Another approach is to set update SQL statements using SQLInsert, SQLUpdate and SQLDelete properties. Use them to specify SQL statements that will be used for corresponding data modifications. It is useful when generating data modification statements is not possible (for example when working with data of a cursor, returned by a stored procedure) or you need to execute some specific statements. You may also assign TOraUpdateSQL component to the UpdateObject property. TOraUpdateSQL component holds all updating SQL statements in one place. You can generate all these SQL statements using ODAC

design time editors. For more careful customization of data update operations you can use InsertObject, ModifyObject and DeleteObject properties of TOraUpdateSQL component.

See Also

- TSmartQuery
- TOraQuery
- TOraStoredProc
- TOraTable
- TOraUpdateSQL

Request Support DAC Forum Provide Feedback

4.2 Connecting in Direct Mode

ODAC Professional Edition allows you to connect to Oracle in two ways: in the Client mode, using Oracle Client software, or in the Direct mode, over TCP/IP. The Direct mode can be enabled using the `TOraSession.Options.Direct` property.

ODAC Connection Modes

By default, ODAC, like most applications that work with Oracle, uses the Oracle Call Interface (OCI) to connect to the Oracle database server. This is referred to as connecting in the Client mode, and is the usual way to develop Oracle applications with a third-generation language. All OCI routines are stored in external libraries, so the executables for applications that work through OCI are small. However, working through OCI requires Oracle client software to be installed on client machines. It is rather inconvenient and causes additional installation and administration expenses. Furthermore, there are some situations where the installation of Oracle client is not advisable or may be even impossible—for example, if you deploy an application to remote machines that are not overseen by a proficient system administrator.

To overcome these challenges, ODAC Professional Edition includes an option to connect to Oracle directly over the network using the TCP/IP protocol. This is referred to as connecting in the `Direct mode`. Connecting in the Direct mode does not require Oracle client software to be installed on client machines. The only requirement for running an application that uses ODAC in the Direct mode, is that the operating system must support the TCP/IP protocol.

Connecting in Direct Mode

To connect to Oracle server in the Direct mode, set up your `TOraSession` instance as follows:

- set the Options.Direct property of your `TOraSession` instance to `True`;
- set the Server property of your `TOraSession` instance, to a string that contains the host address of the database server, port number, and Oracle Service Name or Oracle System Identifier (SID) in the following format:

if you connect to Oracle using Service Name:

`Host:Port/ServiceName`
`Host:Port:sn=ServiceName` (deprecated format)

if you connect to Oracle using SID:

`Host:Port:SID`
`Host:Port:sid=SID` (deprecated format)

Host is the server's IP address or DNS name.

Port is the port number that the server listens to.

SID is a system identifier that specifies the name of an Oracle database instance.

ServiceName is a system alias for an Oracle database instance (or multiple instances).

Note that the syntax used to set up the `Server` property in the Direct mode is different from the Client mode. In the Client mode, this property must be set to the TNS name of the Oracle server.

Note that if the port number is followed by a colon, and the service name prefix (`sn=`) or the SID prefix (`sid=`) is not defined, then by default, the connection will be established using SID.

An example below illustrates the connection to Oracle in the Direct mode. The IP address of the Oracle server is `205.227.44.44`, the port number is `1521` (the most commonly used port for Oracle), and the SID is `orcl` (standard Oracle SID):

```
var
  Session: TOraSession;
. . .
Session.Options.Direct := True;
Session.Username := 'Scott';
Session.Password := 'tiger';
Session.Server := '205.227.44.44:1521:orcl';
```

```
Session.Connect;
```

connecting to Oracle with Service Name:

```
...
Session.Server := '205.227.44.44:1521/orcl';
...
or
...
Session.Server := '205.227.44.44:1521:sn=orcl';
...
```

connecting to Oracle with SID:

```
...
Session.Server := '205.227.44.44:1521:orcl';
...
or
...
Session.Server := '205.227.44.44:1521:sid=orcl';
...
```

This is all you need to do to enable the Direct mode in your application. You do not have to rewrite other parts of your code.

To return to the OCI mode, set `TOraSession.Options.Direct` to `False` and `Session.Server` to the TNS name of your server.

You can connect to Multi-Threaded Server using the Direct mode. The server must be configured to use a specific port and the TTC protocol. This can help you avoid firewall conflicts.

Note: The Direct mode is available in ODAC Professional Edition and Oracle Trial Edition. An attempt to set the `TOraSession.Options.Direct` property to `True` in ODAC Standard Edition will generate a *"Feature is not supported"* error.

Client Mode vs Direct Mode

Applications that use the Client mode and those that use the Direct mode have similar performance and file size. In terms of security, using the Direct mode is the same as using Oracle Client without Oracle Advanced Security. In the Direct mode, ODAC uses DES authentication and does not support Oracle Advanced Security.

Advantages of the Direct mode:

- No need to install and administer Oracle client.
- Reduced system requirements.

Limitations of the Direct mode:

- Only TCP/IP connections supported.
- TOraLoader direct loading is not supported.
- Some issues may occur when using firewalls.
- NLS conversion is not supported on the client side.
- Transparent Application Failover is not supported.
- Change notifications (TOraChangeNotification) are not supported.
- OS Authentication supported for Windows only.

A connection in the Direct mode is managed transparently by an instance of `TOraSession`, and you can easily switch back to OCI in the Client mode at any time if the above limitations become critical to you.

See Also

- TOraSession.Server
- TOraSession.Options
- M:Devart.Odac.TOraSQL.BreakExec()
- TCustomDADataSet.BreakExec

Request Support DAC Forum Provide Feedback

4.3 Master/Detail Relationships

Master/detail (MD) relationship between two tables is a very widespread one. So it is very important to provide an easy way for database application developer to work with it. Lets examine how ODAC implements this feature.

Suppose we have classic MD relationship between "Department" and "Employee" tables.

"Department" table has field Dept_No. Dept_No is a primary key.

"Employee" table has a primary key EmpNo and foreign key Dept_No that binds "Employee" to "Department".

It is necessary to display and edit these tables.

ODAC provides two ways to bind tables. First code example shows how to bind two TCustomOraDataSet components (TOraQuery, TSmartQuery, TOraTable or even TOraStoredProc) into MD relationship via parameters.

```
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TOraQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TOraQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TOraQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee WHERE Dept_No = :Dept_No';
  // connect detail dataset with master via TDataSource component
  MasterSource := TDataSource.Create(Self);
  MasterSource.DataSet := Master;
  Detail.MasterSource := MasterSource;
  // open master dataset and only then detail dataset
  Master.Open;
  Detail.Open;
end;
```

Pay attention to one thing: parameter name in detail dataset SQL must be equal to the field name or the alias in the master dataset that is used as foreign key for detail table. After opening detail dataset always holds records with Dept_No field value equal to the one in the current master dataset record.

There is an additional feature: when inserting new records to detail dataset it automatically fills foreign key fields with values taken from master dataset.

Now suppose that detail table "Department" foreign key field is named DepLink but not

Dept_No. In such case detail dataset described in above code example will not autofill DepLink field with current "Department".Dept_No value on insert. This issue is solved in second code example.

```
procedure TForm1.Form1Create(Sender: TObject);
var
  Master, Detail: TOraQuery;
  MasterSource: TDataSource;
begin
  // create master dataset
  Master := TOraQuery.Create(Self);
  Master.SQL.Text := 'SELECT * FROM Department';
  // create detail dataset
  Detail := TOraQuery.Create(Self);
  Detail.SQL.Text := 'SELECT * FROM Employee';
  // setup MD
  Detail.MasterFields := 'Dept_No';   // primary key in Department
  Detail.DetailFields := 'DepLink'; // foreign key in Employee
  // connect detail dataset with master via TDataSource component
  MasterSource := TDataSource.Create(Self);
  MasterSource.DataSet := Master;
  Detail.MasterSource := MasterSource;
  // open master dataset and only then detail dataset
  Master.Open;
  Detail.Open;
end;
```

In this code example MD relationship is set up using MasterFields and DetailFields properties. Also note that there are no WHERE clause in detail dataset SQL.

To defer refreshing of detail dataset while master dataset navigation you can use DetailDelay option.

Such MD relationship can be local and remote, depending on the TCustomDADataSet.Options.LocalMasterDetail option. If this option is set to True, dataset uses local filtering for establishing master-detail relationship and does not refer to the server. Otherwise detail dataset performs query each time when record is selected in master dataset. Using local MD relationship can reduce server calls number and save server resources. It can be useful for slow connection. CachedUpdates mode can be used for detail dataset only for local MD relationship. Using local MD relationship is not recommended when detail table contains too many rows, because in remote MD relationship only records that correspond to the current record in master dataset are fetched. So, this can decrease network traffic in some cases.

See Also

- TCustomDADataSet.Options
- TMemDataSet.CachedUpdates

4.4 Data Type Mapping

Overview

Data Type Mapping is a flexible and easily customizable gear, which allows mapping between DB types and Delphi field types.

In this article there are several examples, which can be used when working with all supported DBs. In order to clearly display the universality of the Data Type Mapping gear, a separate DB will be used for each example.

Data Type Mapping Rules

In versions where Data Type Mapping was not supported, ODAC automatically set correspondence between the DB data types and Delphi field types. In versions with Data Type Mapping support the correspondence between the DB data types and Delphi field types can be set manually.

Here is the example with the numeric type in the following table of a Oracle database:

```
CREATE TABLE NUMBER_TYPES
(
 ID NUMBER NOT NULL,
 VALUE1 NUMBER(4,0),
 VALUE2 NUMBER(10,0),
 VALUE3 NUMBER(15,0),
 VALUE4 NUMBER(5,2),
 VALUE5 NUMBER(10,4),
 VALUE6 NUMBER(15,6),
 CONSTRAINT PK_NUMBER_TYPES PRIMARY KEY (ID)
)
```

And Data Type Mapping should be used so that:

- the numeric fields with Scale=0 in Delphi would be mapped to one of the field types: TSmallintField, TIntegerField or TlargeintField, depending on Precision
- to save precision, the numeric fields with Precision>=10 and Scale<= 4 would be mapped

to TBCDField

- and the numeric fields with Scale>= 5 would be mapped to TFMTBCDField.

The above in the form of a table:

Oracle data type	Default Delphi field type	Destination Delphi field type
numeric(4,0)	ftFloat	ftSmallint
numeric(10,0)	ftFloat	ftInteger
numeric(15,0)	ftFloat	ftLargeint
numeric(5,2)	ftFloat	ftFloat
numeric(10,4)	ftFloat	ftBCD
numeric(15,6)	ftFloat	ftFMTBCD

To specify that numeric fields with Precision <= 4 and Scale = 0 must be mapped to ftSmallint, such a rule should be set:

```
var
  DBType: Word;
  MinPrecision: Integer;
  MaxPrecision: Integer;
  MinScale: Integer;
  MaxScale: Integer;
  FieldType: TfieldType;
begin
  DBType       := oraNumber;
  MinPrecision := 0;
  MaxPrecision := 4;
  MinScale     := 0;
  MaxScale     := 0;
  FieldType    := ftSmallint;
  OraSession.DataTypeMap.AddDBTypeRule(DBType, MinPrecision, MaxPrecision, M
end;
```

This is an example of the detailed rule setting, and it is made for maximum visualization.Usually, rules are set much shorter, e.g. as follows:

```
  OraSession.DataTypeMap.Clear;
  // rule for numeric(4,0)
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0,      4, 0,     0, ftSma
  // rule for numeric(10,0)
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 5,     10, 0,     0, ftInt
  // rule for numeric(15,0)
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 11, rlAny, 0,     0, ftLar
  // rule for numeric(5,2)
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0,      9, 1, rlAny, ftFlo
  // rule for numeric(10,4)
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 10, rlAny, 1,     4, ftBCD
  // rule for numeric(15,6)
```

```
  OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 10, rlAny, 5, rlAny, ftFMT
```

Rules order

When setting rules, there can occur a situation when two or more rules that contradict to each other are set for one type in the database. In this case, only one rule will be applied — the one, which was set first.

For example, there is a table in an Oracle database:

```
CREATE TABLE PERSON
(
 ID                  NUMBER        NOT NULL,
 FIRSTNAME           VARCHAR2(20)          ,
 LASTNAME            VARCHAR2(30)          ,
 GENDER_CODE         VARCHAR2(1)           ,
 BIRTH_DTTM          DATE                  ,
 CONSTRAINT PK_PERSON PRIMARY KEY (ID)
)
```

TBCDField should be used for NUMBER(10,4), and TFMTBCDField - for NUMBER(15,6) instead of default fields:

Oracle data type	Default Delphi field type	Destination field type
NUMBER(5,2)	ftFloat	ftFloat
NUMBER(10,4)	ftFloat	ftBCD
NUMBER(15,6)	ftFloat	ftFMTBCD

If rules are set in the following way:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0,     9, rlAny, rlAny, ftFl
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny,     0,     4, ftBC
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny,     0, rlAny, ftFM
```

it will lead to the following result:

Oracle data type	Delphi field type
NUMBER(5,2)	ftFloat
NUMBER(10,4)	ftBCD
NUMBER(15,6)	ftFMTBCD

But if rules are set in the following way:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny,     0, rlAny, ftFM
```

```
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny,     0,     4, ftBC
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0,     9, rlAny, rlAny, ftFl
```

it will lead to the following result:

Oracle data type	Delphi field type
NUMBER(5,2)	ftFMTBCD
NUMBER(10,4)	ftFMTBCD
NUMBER(15,6)	ftFMTBCD

This happens because the rule

```
OraSession.DataTypeMap.AddDBTypeRule(oraNumber, 0, rlAny, 0, rlAny, ftFMTBCD
```

will be applied for the NUMBER fields, whose Precision is from 0 to infinity, and Scale is from 0 to infinity too. This condition is met by all NUMBER fields with any Precision and Scale.

When using Data Type Mapping, first matching rule is searched for each type, and it is used for mapping. In the second example, the first set rule appears to be the first matching rule for all three types, and therefore the ftFMTBCD type will be used for all fields in Delphi.

If to go back to the first example, the first matching rule for the NUMBER(5,2) type is the first rule, for NUMBER(10,4) - the second rule, and for NUMBER(15,6) - the third rule. So in the first example, the expected result was obtained.

So it should be remembered that if rules for Data Type Mapping are set so that two or more rules that contradict to each other are set for one type in the database, the rules will be applied in the specifed order.

Defining rules for Connection and Dataset

Data Type Mapping allows setting rules for the whole connection as well as for each DataSet in the application.

For example, such table is created in Oracle:

```
CREATE TABLE PERSON
(
 ID                  NUMBER               NOT NULL  ,
 FIRSTNAME           VARCHAR2(20)       ,
```

```
 LASTNAME          VARCHAR2(30)      ,
 GENDER_CODE       VARCHAR2(1)       ,
 BIRTH_DTTM        DATE         ,
 CONSTRAINT PK_PERSON PRIMARY KEY (ID)
)
```

It is exactly known that the birth_dttm field contains birth day, and this field should be ftDate in Delphi, and not ftDateTime. If such rule is set:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(OraDate, ftDate);
```

all DATETIME fields in Delphi will have the ftDate type, that is incorrect. The ftDate type was expected to be used for the DATETIME type only when working with the person table. In this case, Data Type Mapping should be set not for the whole connection, but for a particular DataSet:

```
OraQuery.DataTypeMap.Clear;
OraQuery.DataTypeMap.AddDBTypeRule(OraDate, ftDate);
```

Or the opposite case. For example, DATETIME is used in the application only for date storage, and only one table stores both date and time. In this case, the following rules setting will be correct:

```
OraSession.DataTypeMap.Clear;
OraSession.DataTypeMap.AddDBTypeRule(OraDate, ftDate);
OraQuery.DataTypeMap.Clear;
OraQuery.DataTypeMap.AddDBTypeRule(OraDate, ftDateTime);
```

In this case, in all DataSets for the DATETIME type fields with the ftDate type will be created, and for OraQuery - with the ftDateTime type.

The point is that the priority of the rules set for the DataSet is higher than the priority of the rules set for the whole connection. This allows both flexible and convenient setting of Data Type Mapping for the whole application. There is no need to set the same rules for each DataSet, all the general rules can be set once for the whole connection. And if a DataSet with an individual Data Type Mapping is necessary, individual rules can be set for it.

Rules for a particular field

Sometimes there is a need to set a rule not for the whole connection, and not for the whole dataset, but only for a particular field.

e.g. there is such table in a Oracle database:

```
CREATE TABLE ITEM
(
 ID NUMBER NOT NULL,
 NAME VARCHAR2(50) NOT NULL,
 GUID VARCHAR2(38),
 CONSTRAINT PK_ITEM PRIMARY KEY (ID)
)
```

The **guid** field contains a unique identifier. For convenient work, this identifier is expected to be mapped to the TGuidField type in Delphi. But there is one problem, if to set the rule like this:

```
OraQuery.DataTypeMap.Clear;
OraQuery.DataTypeMap.AddDBTypeRule(oraVarchar2, ftGuid);
```

then both **name** and **guid** fields will have the ftGuid type in Delphi, that does not correspond to what was planned. In this case, the only way is to use Data Type Mapping for a particular field:

```
OraQuery.DataTypeMap.AddFieldNameRule('GUID', ftGuid);
```

In addition, it is important to remember that setting rules for particular fields has the highest priority. If to set some rule for a particular field, all other rules in the Connection or DataSet will be ignored for this field.

Ignoring conversion errors

Data Type Mapping allows mapping various types, and sometimes there can occur the problem with that the data stored in a DB cannot be converted to the correct data of the Delphi field type specified in rules of Data Type Mapping or vice-versa. In this case, an error will occur, which will inform that the data cannot be mapped to the specified type.

For example:

Database value	Destination field type	Error
'text value'	ftInteger	String cannot be converted to Integer
1000000	ftSmallint	Value is out of range
15,1	ftInteger	Cannot convert float to integer

But when setting rules for Data Type Mapping, there is a possibility to ignore data conversion errors:

```
OraSession.DataTypeMap.AddDBTypeRule(oraVarchar2, ftInteger, True);
```

In this case, the correct conversion is impossible. But because of ignoring data conversion errors, Data Type Mapping tries to return values that can be set to the Delphi fields or DB fields depending on the direction of conversion.

Database value	Destination field type	Result	Result description
'text value'	ftInteger	0	0 will be returned if the text cannot be converted to number
1000000	ftSmallint	32767	32767 is the max value that can be assigned to the Smallint data type
15,1	ftInteger	15	15,1 was truncated to an integer value

Therefore ignoring of conversion errors should be used only if the conversion results are expected.

Request Support DAC Forum Provide Feedback

4.5 Data Encryption

ODAC has built-in algorithms for data encryption and decryption. To enable encryption, you should attach the TCREncryptor component to the dataset, and specify the encrypted fields. When inserting or updating data in the table, information will be encrypted on the client side in accordance with the specified method. Also when reading data from the server, the components decrypt the data in these fields "on the fly".

For encryption, you should specify the data encryption algorithm (the EncryptionAlgorithm property) and password (the Password property). On the basis of the specified password, the key is generated, which encrypts the data. There is also a possibility to set the key directly using the SetKey method.

When storing the encrypted data, in addition to the initial data, you can also store additional information: the GUID and the hash. (The method is specified in the TCREncryptor.DataHeader property).

If data is stored without additional information, it is impossible to determine whether the data is encrypted or not. In this case, only the encrypted data should be stored in the column, otherwise, there will be confusion because of the inability to distinguish the nature of the data. Also in this way, the similar source data will be equivalent in the encrypted form, that is not good from the point of view of the information protection. The advantage of this method is the size of the initial data equal to the size of the encrypted data.

To avoid these problems, it is recommended to store, along with the data, the appropriate GUID, which is necessary for specifying that the value in the record is encrypted and it must be decrypted when reading data. This allows you to avoid confusion and keep in the same column both the encrypted and decrypted data, which is particularly important when using an existing table. Also, when doing in this way, a random initializing vector is generated before the data encryption, which is used for encryption. This allows you to receive different results for the same initial data, which significantly increases security.

The most preferable way is to store the hash data along with the GUID and encrypted information to determine the validity of the data and verify its integrity. In this way, if there was an attempt to falsify the data at any stage of the transmission or data storage, when decrypting the data, there will be a corresponding error generated. For calculating the hash the SHA1 or MD5 algorithms can be used (the HashAlgorithm property).

The disadvantage of the latter two methods - additional memory is required for storage of the auxiliary information.

As the encryption algorithms work with a certain size of the buffer, and when storing the additional information it is necessary to use additional memory, TCREncryptor supports encryption of string or binary fields only (*ftString*, *ftWideString*, *ftBytes*, *ftVarBytes*, *ftBlob*, *ftMemo*, *ftWideMemo*). If encryption of string fields is used, firstly, the data is encrypted, and then the obtained binary data is converted into hexadecimal format. In this case, data storage requires two times more space (one byte = 2 characters in hexadecimal).

Therefore, to have the possibility to encrypt other data types (such as date, number, etc.), it is necessary to create a field of the binary or BLOB type in the table, and then convert it into the desired type on the client side with the help of data mapping.

It should be noted that the search and sorting by encrypted fields become impossible on the server side. Data search for these fields can be performed only on the client after decryption of data using the Locate and LocateEx methods. Sorting is performed by setting the

TMemDataSet.IndexFieldNames property.

Example.

Let's say there is an employee list of an enterprise stored in the table with the following data: full name, date of employment, salary, and photo. We want all these data to be stored in the encrypted form. Write a script for creating the table:

```
CREATE TABLE EMP (
EMPNO NUMBER,
ENAME VARCHAR2(2000),
HIREDATE VARCHAR2(200),
SAL VARCHAR2(200),
FOTO BLOB,
CONSTRAINT PK_EMP PRIMARY KEY (EMPNO)
);
```

As we can see, the fields for storage of the textual information, date, and floating-point number are created with the VARCHAR2 type. This is for the ability to store encrypted information, and in the case of the text field - to improve performance. Write the code to process this information on the client.

```
OraQuery.SQL.Text := 'SELECT * FROM EMP';
OraQuery.Encryption.Encryptor := OraEncryptor;
OraQuery.Encryption.Fields := 'ENAME, HIREDATE, SAL, FOTO';
OraEncryptor.Password := '11111';
OraQuery.DataTypeMap.AddFieldNameRule ('ENAME', ftString);
OraQuery.DataTypeMap.AddFieldNameRule ('HIREDATE', ftDateTime);
OraQuery.DataTypeMap.AddFieldNameRule ('SAL', ftFloat);
OraQuery.Open;
```


4.6 Working in an Unstable Network

The following settings are recommended for working in an unstable network:

```
TCustomDAConnection.Options.LocalFailover = True
TCustomDAConnection.Options.DisconnectedMode = True
TDataSet.CachedUpdates = True
TCustomDADataSet.FetchAll = True
TCustomDADataSet.Options.LocalMasterDetail = True
AutoCommit = True
```

These settings minimize the number of requests to the server. Using

TCustomDAConnection.Options.DisconnectedMode allows DataSet to work without an active connection. It minimizes server resource usage and reduces connection break probability. I. e. in this mode connection automatically closes if it is not required any more. But every explicit operation must be finished explicitly. That means each explicit connect must be followed by explicit disconnect. Read Working with Disconnected Mode topic for more information.

Setting the FetchAll property to True allows to fetch all data after cursor opening and to close connection. If you are using master/detail relationship, we recommend to set the LocalMasterDetail option to True.

It is not recommended to prepare queries explicitly. Use the CachedUpdates mode for DataSet data editing. Use the TCustomDADataSet.Options.UpdateBatchSize property to reduce the number of requests to the server.

If a connection breaks, a fatal error occurs, and the OnConnectionLost event will be raised if the following conditions are fulfilled:

- There are no active transactions;
- There are no opened and not fetched datasets;
- There are no explicitly prepared datasets or SQLs.

If the user does not refuse suggested RetryMode parameter value (or does not use the OnConnectionLost event handler), ODAC can implicitly perform the following operations:

```
Connect;
DataSet.ApplyUpdates;
DataSet.Open;
```

I.e. when the connection breaks, implicit reconnect is performed and the corresponding operation is reexecuted. We recommend to wrap other operations in transactions and fulfill their reexecuting yourself.

The using of Pooling in Disconnected Mode allows to speed up most of the operations because of connecting duration reducing.

See Also

- FailOver demo
- Working with Disconnected Mode

- TCustomDAConnection.Options
- TCustomDAConnection.Pooling

 Request Support DAC Forum Provide Feedback

4.7 Secure Connections

This section describes how to establish a secure connection between a client application and Oracle Database.

- Connecting via SSL
- Connecting via SSH
- HTTP/HTTPS Network Tunneling

 Request Support DAC Forum Provide Feedback

4.7.1 Connecting via SSL

Connecting to Oracle Database Using SSL

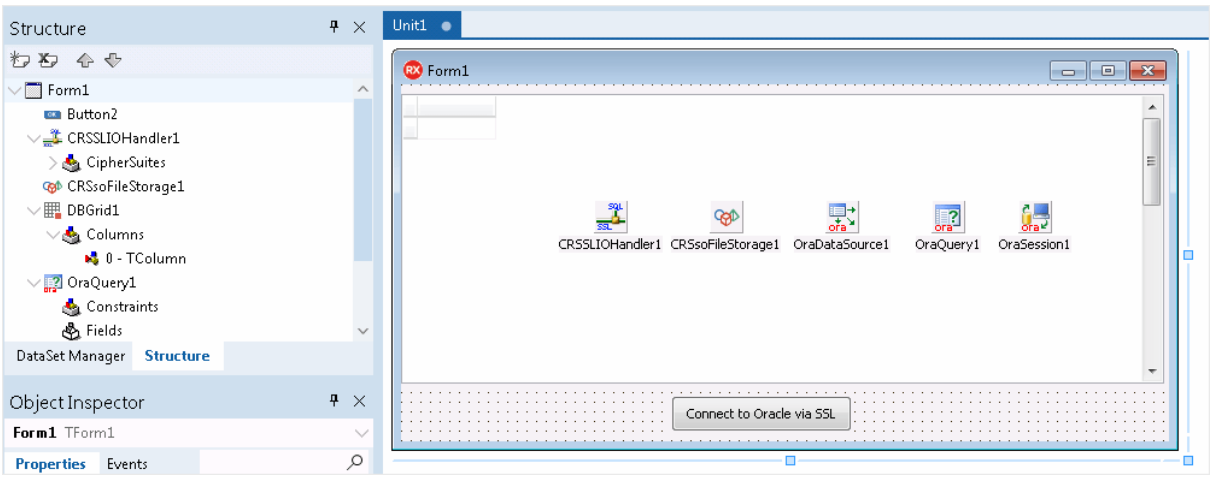
This section discusses how to connect a client application to Oracle Database using SSL (Secure Sockets Layer), which is an industry standard protocol for secure access to a remote machine over untrusted networks. It runs on top of TCP/IP to secure client-server communications by allowing an SSL-enabled client to authenticate itself to an SSL-enabled server and vice versa. During server authentication, an SSL-enabled client application uses standard techniques of public-key cryptography to verify the server's identity by checking that the server's certificate is issued by a trusted certificate authority (CA) and proves the ownership of the public key.

Conversely, SSL client authentication allows the server to validate the client's identity. The client and server can also authenticate each other using self-signed certificates, however, you will almost never want to use a self-signed certificate, except for an Intranet or a development server. After establishing an SSL connection, the client and server can exchange messages that are symmetrically encrypted with the shared secret key. SSL is the recommended method to establish a secure connection to Oracle due to easier configuration and higher performance, compared to SSH.

To establish an SSL connection to the server with ODAC, you must compile and install the `TCRSSLIOHandler` component, which is distributed with SecureBridge and is required to bind ODAC with SecureBridge. The installation instructions for the component are provided in the `Readme.html` file, which is located by default in "My Documents\Devart\ODAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge".

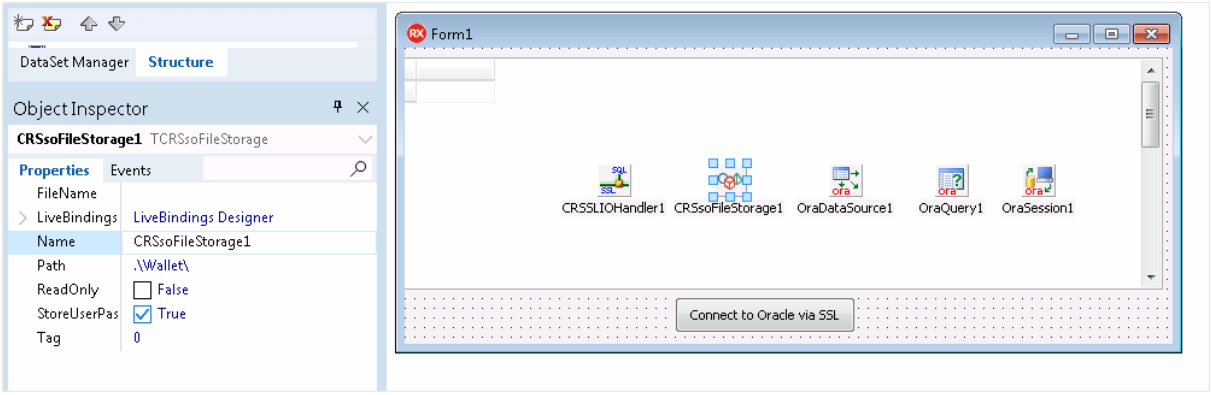
Connecting to Oracle Database Using Oracle Wallet

1. Place the following components on the form: `TOraSession`, `TOraQuery`, `TOraDataSource`, `TDBGrid`, `TButton`, `TCRSSLIOHandler`, `TCRSsoFileStorage`.

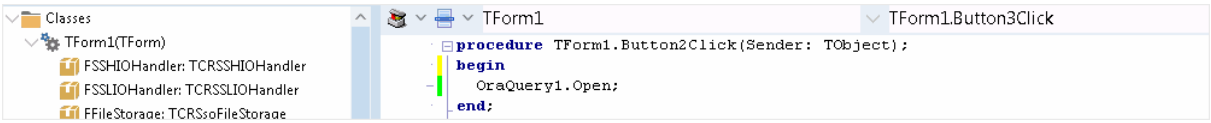


2. Select the `TOraSession` component and set the `IOHandler` property to an instance of `TCRSSLIOHandler`.
3. Select the `TDBGrid` component and set the `DataSource` property to an instance of `TOraDataSource`.
4. Select the `TOraDataSource` component and set the `DataSet` property to an instance of `TOraQuery`.
5. Select the `TOraQuery` component and set the `Session` property to an instance of `TOraSession`.
6. Double-click the `TOraQuery` component and specify a SQL query to execute against Oracle Database.
7. Select the `TCRSsoFileStorage` component and specify the path to the wallet file. A wallet is container for storing authentication and signing credentials, including keys and certificates

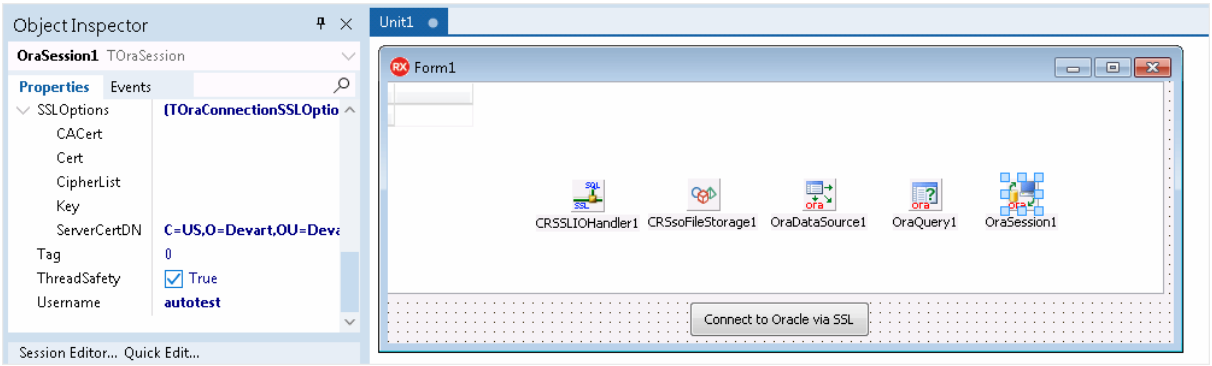
needed by SSL. See this document for information on creating an Oracle wallet. If you are using Oracle Cloud, see this document for information on obtaining wallet files.



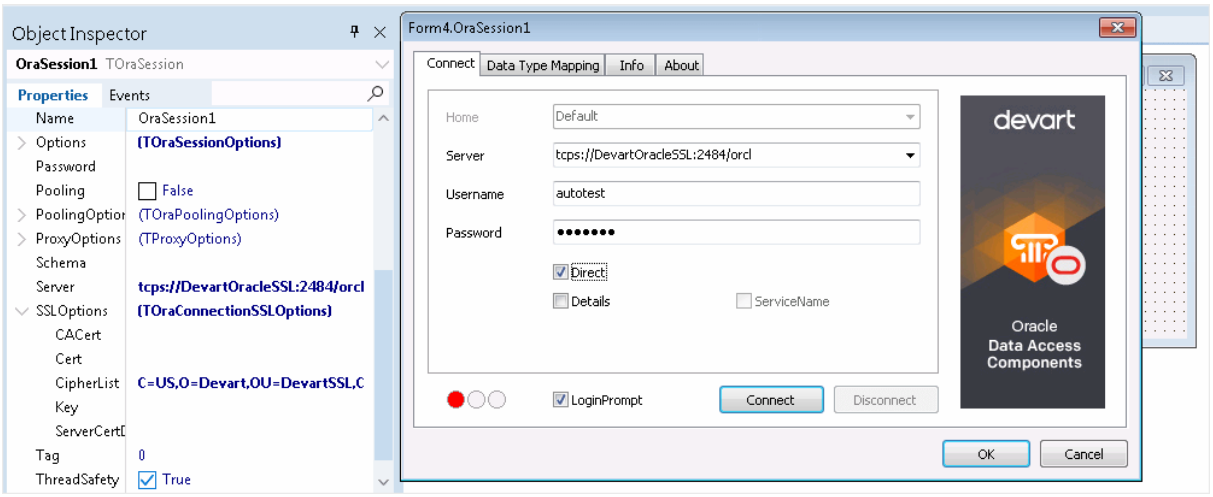
8. Select the `TCRSSLIOHandler` component and set the `Storage` property to an instance of `TCRSsoFileStorage`.
9. Select the `TButton` component and create an `OnClick` event. Add the code to call the `Open` method of `TOraQuery` when the button is clicked.



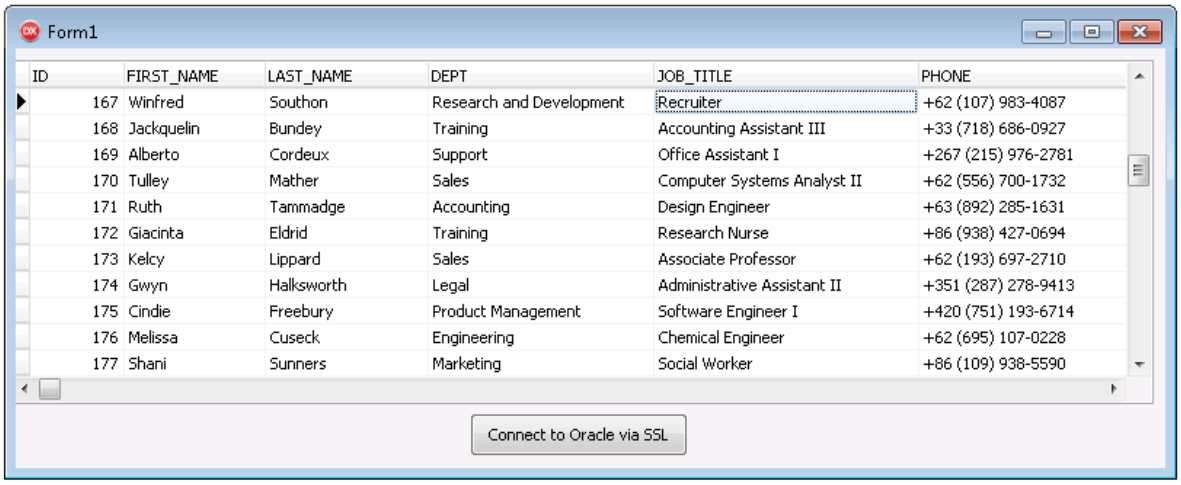
10. Select the `TOraSession` component and specify the server's distinguished name (DN) in the `ServerCertDN` property of `SSLOptions` to enable server DN matching. It is used to check whether the server is genuine by matching the server's global database name against the DN from the server certificate. See this document for information on editing the client network configuration files.



11.Double-click the `TOraSession` component and specify the server address, port, username and password.



12.Compile and run the application.

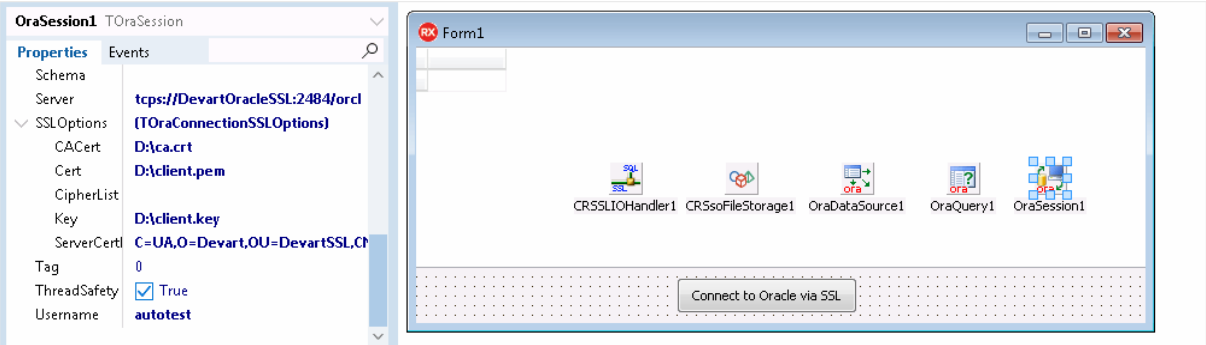


Connecting to Oracle Database Using SSL Certificates and Keys

The steps are similar to the above, except that you specify the server and client SSL certificates and the private client key instead of wallet files, thus you do not need the `TCRSsoFileStorage` component.

Select the `TOraSession` component and expand `SSLOptions`. Specify the server certificate in the `CACert` property, the client certificate in the `Cert` property, the private client key in the `Key`

property and the server's distinguished name (DN) in the `ServerCertDN` property.



 Request Support DAC Forum Provide Feedback

4.7.2 Connecting via SSH

Connecting to Oracle Database Through SSH

This section dicusses how to connect a client application to Oracle Database through SSH. SSH is a network protocol for secure remote login to another system over the Internet by connecting the SSH client to the SSH server. SSH provides a mechanism for establishing a secure connection between the client and the remote server, which authenticate each other and exchange messages. It employs different forms of symmetrical encryption, asymmetrical encryption, and hashing. The SSH client initiates a connection and uses public key cryptography to verify the identity of the SSH server.

It is possible to use SSH an an encryption method to secure the connection between a Delphi application and an Oracle server. You can embed the SSH client functionality into your application and install the SSH server on a remote machine where your Oracle server resides. The SSH client connects to the SSH server, which sends all commands to Oracle Database.

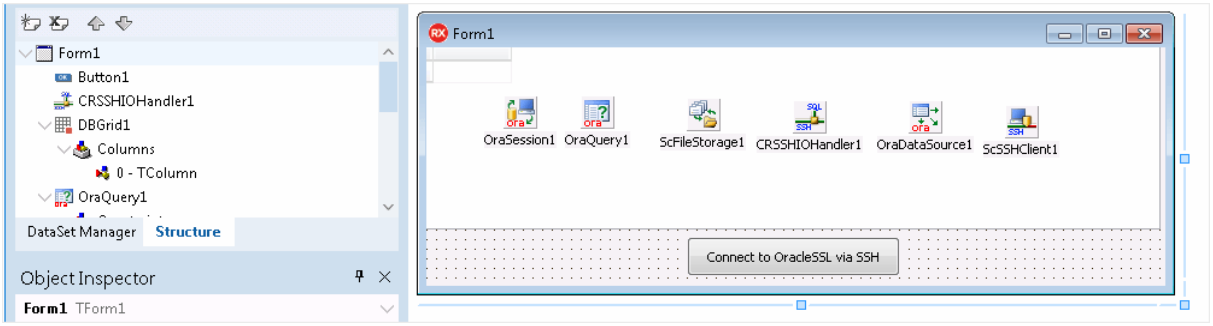
SSH key-based authentication is done by public and private keys that a client uses to authenticate itself when logging into an SSH server. The server key is used is used by the client to authenticate the SSH server and is specified in the TScSSHClient.HostKeyName property. The client key is used by the SSH server to authenticate the client and is specified in the TScSSHClient.PrivateKeyName property. Note that the private key contains the public key. See SecureBridge tutorial on configuring the SSH server.

The SSH server is required to replicate the steps in this tutorial and encrypt the network connection between the client application and Oracle Database. You can build the SSH server demo project, which is distributed with SecureBridge ("Documents\Devart\ODAC for RAD Studio\Demos\TechnologySpecific\SecureBridge\Demo\SSH") and run the executable file.

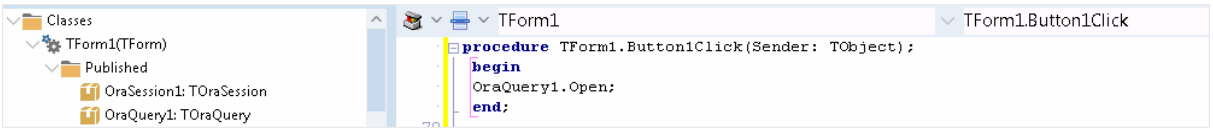
To establish a connection to a remote SSH server, you must compile and install the `TCRSSHIOHandler` component, which is distributed with SecureBridge and is required to bind ODAC with SecureBridge. The installation instructions for the component are provided in the Readme.html file, which is located by default in "My Documents\Devart\ODAC for RAD Studio XX\Demos\TechnologySpecific\SecureBridge".

Sample Application That Connects to Oracle Database Through SSH

1. Place the following components on the form: `TOraSession`, `TOraQuery`, `TOraDataSource`, `TDBGrid`, `TButton`, `TCRSSHIOHandler`, `TScSSHClient`, and `TScFileStorage`.

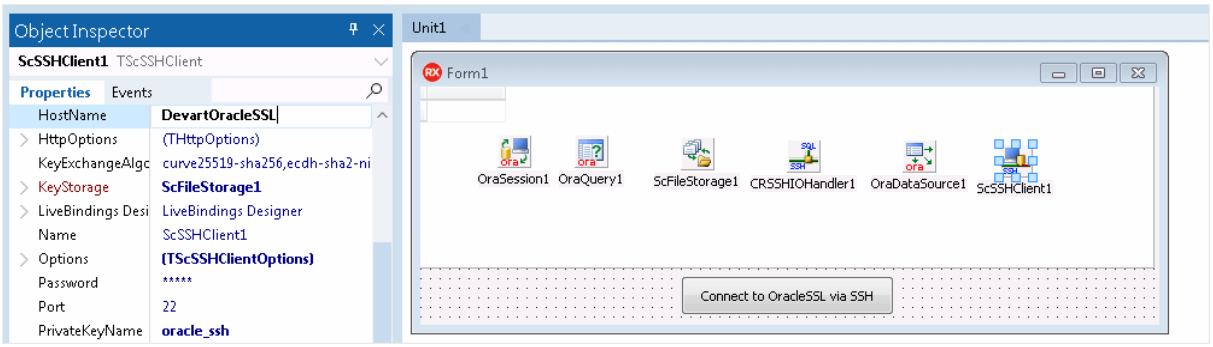


2. Select the `TDBGrid` component and set the `DataSource` property to an instance `TOraDataSource`.
3. Select the `TOraDataSource` component and set the `DataSet` property to an instance of `TOraQuery`.
4. Select the `TOraQuery` component and set the `Session` property to an instance of `TOraSession`. Double-click `TOraQuery` and specify a SQL query to execute against Oracle Database.
5. Select the `TButton` component and create an `OnClick` event. Add the code to call the `Open` method of `TOraQuery` when the button is clicked.

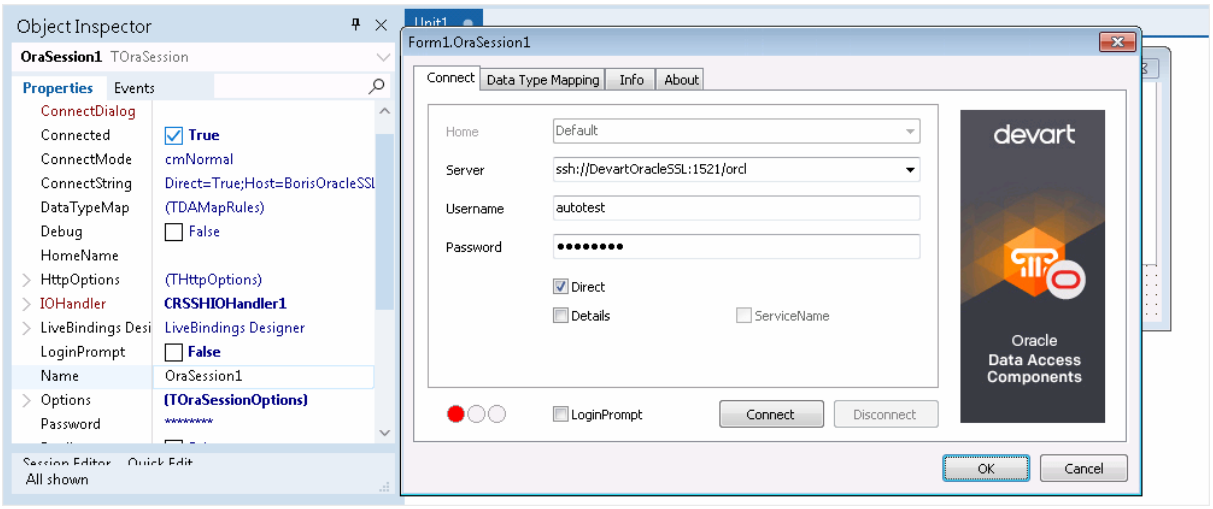


6. Select the `TCRSSHIOHandler` component and set the `Client` property to `TScSSHClient`.
7. Select the `TScFileStorage` component and specify the directory for storing information about keys and users in the `Path` property. Follow the instructions to generate a pair of keys for authenticating the server by the client.
8. Select the `TScSSHClient` component and specify the server public key in the `HostKeyName` property and the client private key in the `PrivateKeyName` property. Specify the address of the SSH server in the `HostName` property and the port, user, and password in corresponding properties. Set the `KeyStorage` property to an instance of `TScFileStorage`.

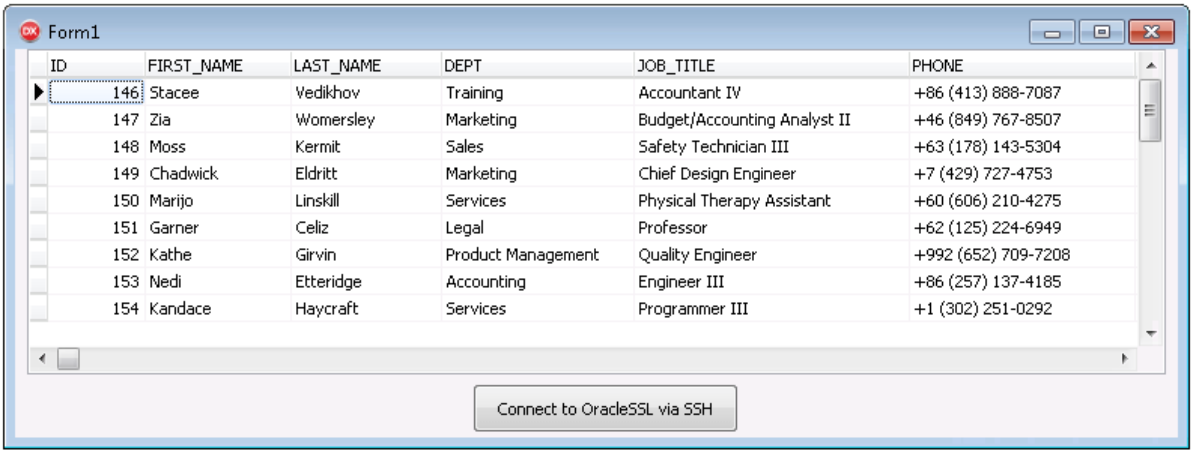
   If you are connecting to `Oracle Cloud`, leave the `Password` and `HostKeyName` properties empty — only specify `PrivateKeyName` and `User` (the default username is `opc`). See the Oracle documentation for information on generating SSH keys.



9. Select the `TOraSession` component and set the `IOHandler` property to an instance of `TCRSSHIOHandler`. Double-click `TOraSession` and specify the server address, port, service name, and user credentials.



10.Compile and run the application.



It is not mandatory to use the `TScSSHServer` component as the SSH server — you can use any other server that implements the SSH protocol.

4.7.3 Network Tunneling

Connecting to Oracle Database Through HTTP Tunnel

This section discusses how to connect a client application to Oracle Database in two ways: directly and through an HTTP tunnel. If you need to connect to Oracle Database in conditions

of restricted connectivity, e.g. when a database server is hidden behind a firewall, or you need to transmit private network data through a public network, you can set up an HTTP tunnel to create a direct network link between two locations. The tunnel is created by an intermediary called a proxy server.

Direct Connection

Direct connection implies that a client connects to a server through a directly connected network, without IP routing: you only need to specify the server address, port number, service name, and user credentials. This is also the fastest and preferred way to communicate with an Oracle server.

Code sample for a direct connection:

```
var
  OraSession: TOraSession;
...
OraSession := TOraSession.Create(self);
OraSession.Options.Direct := True;
OraSession.Server := '205.227.44.44:1521/ORCL1020';
OraSession.Username := 'Scott';
OraSession.Password := 'Tiger';
OraSession.Connect;
```

Connecting Through HTTP Tunnel

When an Oracle server is hidden behind a firewall, the client is not able to connect to the server directly on a specified port. If your firewall allows HTTP connections, you can use ODAC with a properly configured web server to connect to the database server. ODAC supports HTTP tunneling based on the PHP script.

A possible scenario of using HTTP tunneling: the client needs to access the database of a website from a remote machine, but access to the designated port of the database server is forbidden - only connections on the HTTP port 80 are allowed. To establish a connection in this scenario, you must deploy the tunnel.php script, which is distributed with the provider

package, on the web server. It enables access to the database server through an HTTP tunnel. The script must be accessible through HTTP. You can verify script accessibility using any web browser. The script file is located in the HTTP folder of the installed provider: `"%Program Files%\Devart\ODAC for RAD Studio XX\HTTP\tunnel.php"`. The only requirement to the server is support for PHP 5.

To connect to the database, you must set the `TOraSession` parameters as you do for a direct connection, then set the `HttpOptions.Enabled` property to `True`, and set the following parameters, specific to the HTTP tunneling:

Property	Mandatory	Meaning
HttpOptions.Url	Yes	The URL of the PHP script for HTTP tunneling. For example, if the script is is located in the root directory, the URL may look like this: `https://host/tunnel.php`.
HttpOptions.Username, HttpOptions.Password	No	The username and password for the password-protected directory that contains the HTTP tunneling script.

Connecting Through Proxy and HTTP Tunnel

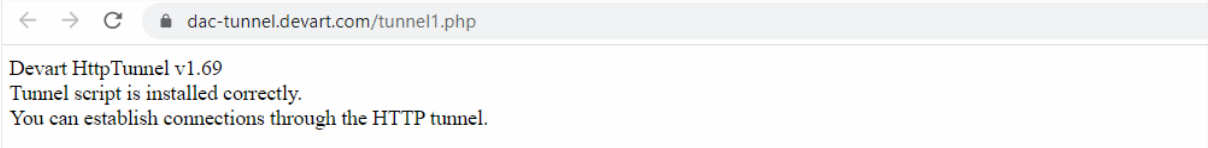
The HTTP tunneling server may be not be directly accessible from the client machine, for example, the client address is `10.0.0.2` and the server address is `205.227.44.44:1521/ORCL1020`. The client and server reside in different networks, so the client can only reach it through the proxy server at `10.0.0.1`, which listens on port `808`. In this case, in addition to `TOraSession.HttpOptions`, you have to set values for `HttpOptions.ProxyOptions`, for example:

```
var
  OraSession: TOraSession;
...
OraSession := TOraSession.Create(self);
OraSession.Options.Direct := True;
OraSession.Server := '205.227.44.44:1521/ORCL1020';
OraSession.Username := 'Scott';
OraSession.Password := 'Tiger';
OraSession.HttpOptions.Enabled := True;
OraSession.HttpOptions.Url := 'https://dac-tunnel.devart.com/tunnel1.php';
OraSession.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';
OraSession.HttpOptions.ProxyOptions.Port := 808;
OraSession.HttpOptions.ProxyOptions.Username := 'ProxyUser';
OraSession.HttpOptions.ProxyOptions.Password := 'ProxyPassword';
OraSession.Connect;
```

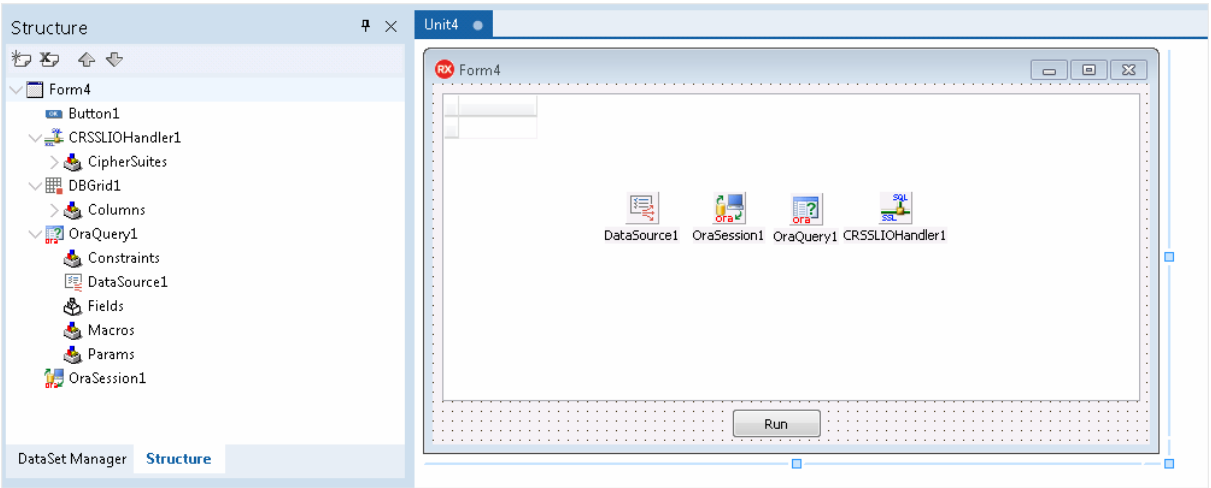
Note that setting the parameters for `OraSession.HttpOptions.ProxyOptions` automatically enables the use of the proxy server.

Sample Aplication That Uses HTTP/HTTPS Tunneling

1. Open your browser and visit the URL of the `tunnel.php` script on your server to verify that the script has been properly installed.

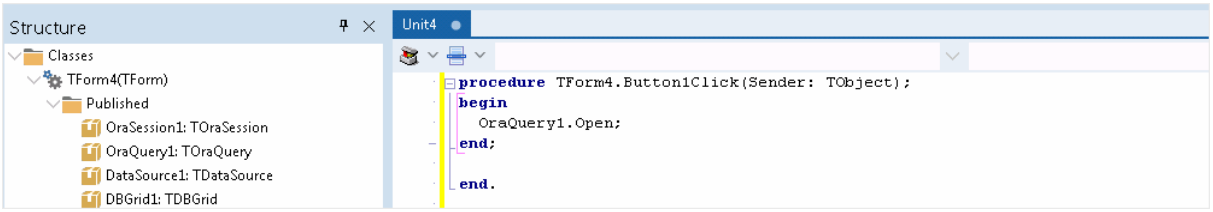


2. Run RAD Studio and create a new VCL application.
3. Place the following components on the form: `TOraSession`, `TOraQuery`, `TDataSource`, `TDBGrid`, `TButton`, and `TCRSSLIOHandler`. The last component is required when connecting through HTTPS. `TCRSSLIOHandler` is distributed with SecureBridge and is required for binding ODAC with SecureBridge. The installation instructions for the component are provided in `Readme.html`, which is located by default in `"My Documents\Devart\ODAC for RAD Studio XX \Demos\TechnologySpecific\SecureBridge\DelphiXX."`



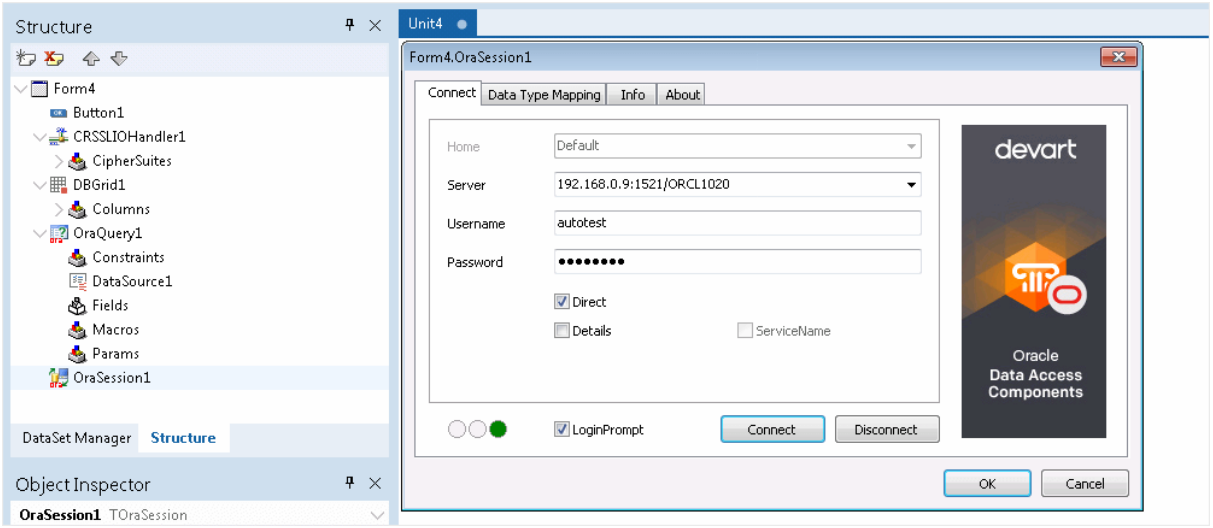
4. Select `TDBGrid` and set the `DataSource` property to an instance of `TDataSource`.
5. Select the `TDataSource` component and set the `DataSet` property to an instance of `TOraQuery`.

6. Select `TOraQuery` and set the `Session` property to an instance of `TOraSession`. Double-click the component and enter an SQL statement to be executed against Oracle Database.
7. Double-click `TButton` to switch to the code view. Add the code to call the `Open` method of `TOraQuery` when the button is clicked.

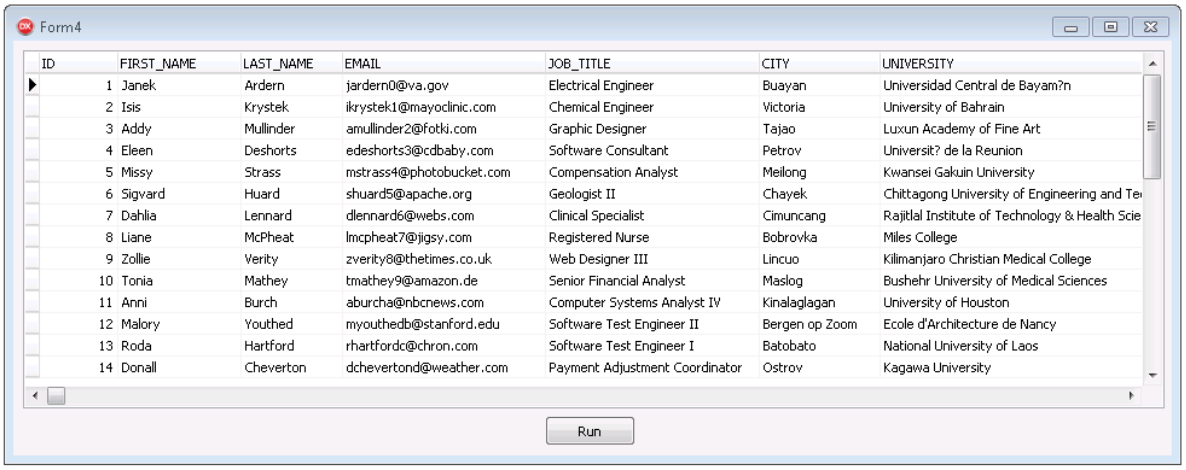


8. Select the `TOraSession` component. If you use an HTTPS tunnel, set the `IOHandler` property to `CRSSLIOHandler1`. Expand the `HttpOptions` and enter the URL of the `tunnel.php` script on your server.

9. Double-click the `TOraSession` component. Specify your server address, port, service name, username and password for the Oracle user. Click `Connect` to test connection to the Oracle server.



10.Press `F9` to compile and run the project, and click the button to run the query against the database through HTTPS and display the data in the form.



Additional Information

There is one more way to tunnel network traffic. The Secure Shell forwarding, or SSH, can be used for data forwarding. HHowever, SSH is designed to encrypt traffic rather than traverse firewalls. The Connecting via SSH document describes how to set up an SSH connection in ODAC.

Keep in mind that traffic tunneling or encryption always increases the CPU usage and bandwidth utilization. It is recommended that you use direct connection whenever possible.

4.8 Disconnected Mode

In disconnected mode a connection opens only when it is required. After performing all server calls connection closes automatically until next server call is required. Datasets remain opened when connection closes. Disconnected Mode may be useful for saving server resources and operating in an unstable or expensive network. Drawback of using disconnected mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down application work. We recommend to use pooling to solve this problem. For additional information see TCustomDAConnection.Pooling.

To enable disconnected mode set TCustomDAConnection.Options.DisconnectedMode to True.

In disconnected mode a connection is opened for executing requests to the server (if it was not opened already) and is closed automatically if it is not required any more. If the connection was explicitly opened (the Connect method was called or the Connected property was explicitly set to True), it does not close until the Disonnect method is called or the Connected property is set to False explicitly.

The following settings are recommended to use for working in disconnected mode:

```
TDataSet.CachedUpdates = True
TCustomDADataSet.FetchAll = True
TCustomDADataSet.Options.LocalMasterDetail = True
AutoCommit = True
```

These settings minimize the number of requests to the server.

Disconnected mode features

If you perform a query with the FetchAll option set to True, connection closes when all data is fetched if it is not used by someone else. If the FetchAll option is set to false, connection does not close until all data blocks are fetched.

If explicit transaction was started, connection does not close until the transaction is committed or rolled back.

If the query was prepared explicitly, connection does not close until the query is unprepared or

its SQL text is changed.

If dataset uses TCustomDADataSet.LockMode set to lmLockImmediate, connection opens when user starts to edit the record, and closes after user posts or cancels changes.

See Also

- TCustomDAConnection.Options
- FetchAll
- Devart.Odac.TOraQuery.LockMode
- TCustomDAConnection.Pooling
- TCustomDAConnection.Connect
- TCustomDAConnection.Disonnect
- Connected
- Working in unstable network

Request Support DAC Forum Provide Feedback

4.9 Batch Operations

Data amount processed by modern databases grows steadily. In this regard, there is an acute problem – database performance. Insert, Update and Delete operations have to be performed as fast as possible. Therefore Devart provides several solutions to speed up processing of huge amounts of data. So, for example, insertion of a large portion of data to a DB is supported in the TOraLoader. Unfortunately, TOraLoader allows to insert data only – it can't be used for updating and deleting data.

The new version of Devart Delphi Data Access Components introduces the new mechanism for large data processing — Batch Operations. The point is that just one parametrized Modify SQL query is executed. The plurality of changes is due to the fact that parameters of such a query will be not single values, but a full array of values. Such approach increases the speed of data operations dramatically. Moreover, in contrast to using TOraLoader, Batch operations can be used not only for insertion, but for modification and deletion as well.

Let's have a better look at capabilities of Batch operations with an example of the

BATCH_TEST table containing attributes of the most popular data types.

Batch_Test table generating script

```
CREATE TABLE BATCH_TEST
(
  ID    INTEGER NOT NULL PRIMARY KEY,
  F_INTEGER INTEGER,
  F_FLOAT FLOAT,
  F_STRING  VARCHAR2(250),
  F_DATE  DATE
)
```

Batch operations execution

To insert records into the BATCH_TEST table, we use the following SQL query:

```
 INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_FLOAT, :F_STRING, :F_DAT
```

When a simple insertion operation is used, the query parameter values look as follows:

Parameters				
:ID	:F_INTEGER	:F_FLOAT	:F_STRING	:F_DATE
1	100	2.5	‘String Value 1'	01.09.2015

After the query execution, one record will be inserted into the BATCH_TEST table.

When using Batch operations, the query and its parameters remain unchanged. However, parameter values will be enclosed in an array:

Parameters				
:ID	:F_INTEGER	:F_FLOAT	:F_STRING	:F_DATE
1	100	2.5	‘String Value 1'	01.09.2015
2	200	3.15	‘String Value 2'	01.01.2000
3	300	5.08	‘String Value 3'	09.09.2010
4	400	7.5343	‘String Value 4'	10.10.2015
5	500	0.4555	‘String Value 5'	01.09.2015

Now, 5 records are inserted into the table at a time on query execution.

How to implement a Batch operation in the code?

Batch INSERT operation sample

Let’s try to insert 1000 rows to the BATCH_TEST table using a Batch Insert operation:

```
var
  i: Integer;
begin
  // describe the SQL query
  OraQuery1.SQL.Text := 'INSERT INTO BATCH_TEST VALUES (:ID, :F_INTEGER, :F_
  // define the parameter types passed to the query :
  OraQuery1.Params[0].DataType := ftInteger;
  OraQuery1.Params[1].DataType := ftInteger;
  OraQuery1.Params[2].DataType := ftFloat;
  OraQuery1.Params[3].DataType := ftString;
  OraQuery1.Params[4].DataType := ftDateTime;
  // specify the array dimension:
  Query1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to Query1.Params.ValueCount - 1 do begin
    OraQuery1.Params[0][i].AsInteger := i + 1;
    OraQuery1.Params[1][i].AsInteger := i + 2000 + 1;
    OraQuery1.Params[2][i].AsFloat := (i + 1) / 12;
    OraQuery1.Params[3][i].AsString := 'Values ' + IntToStr(i + 1);
    OraQuery1.Params[4][i].AsDateTime := Now;
  end;
  // insert 1000 rows into the BATCH_TEST table
  OraQuery1.Execute(1000);
end;
```

This command will insert 1000 rows to the table with one SQL query using the prepared array of parameter values. The number of inserted rows is defined in the Iters parameter of the Execute(Iters: integer; Offset: integer = 0) method. In addition, you can pass another parameter – Offset (0 by default) – to the method. The Offset parameter points the array element, which the Batch operation starts from.

We can insert 1000 records into the BATCH_TEST table in 2 ways.

All 1000 rows at a time:

```
Query1.Execute(1000);
```

2×500 rows:

```
// insert first 500 rows
OraQuery1.Execute(500, 0);
// insert next 500 rows
OraQuery1.Execute(500, 500);
```

500 rows, then 300, and finally 200:

```
// insert 500 rows
Query1.Execute(500, 0);
// insert next 300 rows starting from 500
OraQuery1.Execute(300, 500);
// insert next 200 rows starting from 800
Query1.Execute(200, 800);
```

Batch UPDATE operation sample

With Batch operations we can modify all 1000 rows of our BATCH_TEST table just this simple:

```
var
  i: Integer;
begin
  // describe the SQL query
  Query1.SQL.Text := 'UPDATE BATCH_TEST SET F_INTEGER=:F_INTEGER, F_FLOAT=:F
  // define parameter types passed to the query:
  Query1.Params[0].DataType := ftInteger;
  Query1.Params[1].DataType := ftFloat;
  Query1.Params[2].DataType := ftString;
  Query1.Params[3].DataType := ftDateTime;
  Query1.Params[4].DataType := ftInteger;
  // specify the array dimension:
  Query1.Params.ValueCount := 1000;
  // populate the array with parameter values:
  for i := 0 to 1000 - 1 do begin
    Query1.Params[0][i].AsInteger := i - 2000 + 1;
    Query1.Params[1][i].AsFloat := (i + 1) / 100;
    Query1.Params[2][i].AsString := 'New Values ' + IntToStr(i + 1);
    Query1.Params[3][i].AsDateTime := Now;
    Query1.Params[4][i].AsInteger := i + 1;
  end;
  // update 1000 rows in the BATCH_TEST table
  Query1.Execute(1000);
end;
```

Batch DELETE operation sample

Deleting 1000 rows from the BATCH_TEST table looks like the following operation:

```
var
  i: Integer;
begin
  // describe the SQL query
  OraQuery1.SQL.Text := 'DELETE FROM BATCH_TEST WHERE ID=:ID';
  // define parameter types passed to the query:
  OraQuery1.Params[0].DataType := ftInteger;
  // specify the array dimension
  OraQuery1.Params.ValueCount := 1000;
  // populate the arrays with parameter values
  for i := 0 to 1000 - 1 do
    OraQuery1.Params[0][i].AsInteger := i + 1;
  // delete 1000 rows from the BATCH_TEST table
  OraQuery1.Execute(1000);
end;
```

Performance comparison

The example with BATCH_TEST table allows to analyze execution speed of normal

operations with a database and Batch operations:

Operation Type	25 000 records	
	Standard Operation (sec.)	Batch Operation (sec.)
Insert	17.64	0.59
Update	18.28	1.20
Delete	16.19	0.45
		The less, the better.

It should be noted, that the retrieved results may differ when modifying the same table on different database servers. This is due to the fact that operations execution speed may differ depending on the settings of a particular server, its current workload, throughput, network connection, etc.

Thing you shouldn't do when accessing parameters in Batch operations!

When populating the array and inserting records, we accessed query parameters by index. It would be more obvious to access parameters by name:

```
for i := 0 to 9999 do begin
  OraQuery1.Params.ParamByName('ID')[i].AsInteger := i + 1;
  OraQuery1.Params.ParamByName('F_INTEGER')[i].AsInteger := i + 2000 + 1;
  OraQuery1.Params.ParamByName('F_FLOAT')[i].AsFloat := (i + 1) / 12;
  OraQuery1.Params.ParamByName('F_STRING')[i].AsString := 'Values ' + IntToS
  OraQuery1.Params.ParamByName('F_DATE')[i].AsDateTime := Now;
end;
```

However, the parameter array would be populated slower, since you would have to define the ordinal number of each parameter by its name in each loop iteration. If a loop is executed 10000 times – **performance loss can become quite significant**.

Request Support DAC Forum Provide Feedback

4.10 Increasing Performance

This topic considers basic stages of working with DataSet and ways to increase performance on each of these stages.

Connect

If your application performs Connect/Disconnect operations frequently, additional performance can be gained using pooling mode (TCustomDAConnection.Pooling = True). It reduces connection reopening time greatly (hundreds times). Such situation usually occurs in web applications.

Execute

If your application executes the same query several times, you can use the TCustomDADataSet.Prepare method or set the TDADataSetOptions.AutoPrepare property to increase performance. For example, it can be enabled for Detail dataset in Master/Detail relationship or for update objects in TCustomDAUpdateSQL. The performance gain achieved this way can be anywhere from several percent to several times, depending on the situation.

To execute SQL statements a TOraSQL component is more preferable than TOraQuery. It can give several additional percents performance gain.

Using DML Array parameters, combined with query preparing can give a considerable performance gain. For example if you need to perform an insert query plenty of times, it is recommended to do it the following way to get the best performance. At first, set parameter data types, and then prepare the query. After that, set param Lengths and fill them with values. Finally, execute the query. For instance:

```
OraSQL1.ParamByName('Param1').DataType := ftInteger;
OraSQL1.Prepare;
OraSQL1.ParamByName('Param1').Param[0].Length := 1000;
for i := 1 to 1000
  OraSQL1.ParamByName('Param1').Param[0].ItemAsInteger[i] := 123;
OraSQL1.Execute;
```

If you execute many different SELECT statements, you can gain additional performance by setting the TOraSQL.StatementCache property to True. This feature is available only with Oracle 9.2i and higher. The TOraSession.Options.StatementCache property should be set to True to use this feature. But using the StatementCache property you may easy step over maximum open cursors on Oracle server. And thus you must be attentive when using this option.

If the TCustomDADataSet.Options.StrictUpdate option is set to False, the RowsAffected property is not calculated and becomes equal zero. This can improve performance of query executing, so if you need to execute many data updating statements at once and you don't mind affected rows count, set this option to False.

Fetch

In some situations you can increase performance a bit by using TCustomDADataSet.Options.CompressBlobMode. Sometimes using TOraDataSet.Options.DeferredLobRead can give some additional performance, because Lobs will be read when they are required. You can also set TOraSession.Options.OptimizerMode to adjust the fetch performance.

Oracle optimizer can be tuned to increase performance of SQL statements executing and data fetching.

You can also tweak your application performance by using the following properties of TCustomDADataSet descendants:

- FetchRows
- Options.FlatBuffers
- Options.LongStrings
- UniDirectional

See the descriptions of these properties for more details and recommendations.

Navigate

The Locate function works faster when dataset is locally sorted on KeyFields fields. Local dataset sorting can be set with the IndexFieldNames property. Performance gain can be large if the dataset contains a large number of rows.

Lookup fields work faster when lookup dataset is locally sorted on lookup Keys.

Setting the TDADataSetOptions.CacheCalcFields property can improve performance when locally sorting and locating on calculated and lookup fields. It can be also useful when calculated field expressions contain complicated calculations.

Setting the TDADataSetOptions.LocalMasterDetail option can improve performance greatly by avoiding server requests on detail refreshes. Setting the TDADataSetOptions.DetailDelay option can be useful for avoiding detail refreshes when switching master DataSet records frequently.

Update

If your application updates datasets in the CachedUpdates mode, then setting the TCustomDADataSet.Options.UpdateBatchSize option to more than 1 can improve performance several hundred times more by reducing the number of requests to the server.

Specifying update SQL statements in a dataset improves performance because of omitting SQL statements generation and automatic preparation of internal updating datasets that are created for every kind of update SQL statements.

You can also increase the data sending performance a bit (several percents) by using Dataset.UpdateObject.ModifyObject, Dataset.UpdateObject, etc. Little additional performance improvement can be reached by setting the AutoPrepare property for these objects.

Insert

If you are about to insert a large number of records into a table, you should use the T:Devart.Odac.TOraLoader component instead of Insert/Post methods, or execution of the INSERT commands multiple times in a cycle. Sometimes usage of T:Devart.Odac.TOraLoader improves performance several times.

4.11 Macros

Macros help you to change SQL statements dynamically. They allow partial replacement of the query statement by user-defined text. Macros are identified by their names which are then referred from SQL statement to replace their occurrences for associated values.

First step is to assign macros with their names and values to a dataset object.

Then modify SQL statement to include macro names into desired insertion points. Prefix each name with & ("at") sign to let ODAC discriminate them at parse time. Resolved SQL statement will hold macro values instead of their names but at the right places of their occurrences. For example, having the following statement with the TableName macro name:

```
SELECT * FROM &TableName
```

You may later assign any actual table name to the macro value property leaving your SQL statement intact.

```
Query1.SQL.Text := 'SELECT * FROM &TableName';
Query1.MacroByName('TableName').Value := 'Dept';
```

```
Query1.Open;
```

ODAC replaces all macro names with their values and sends SQL statement to the server when SQL execution is requested.

Note that there is a difference between using TMacro AsString and Value properties. If you set macro with the AsString property, it will be quoted. For example, the following statements will result in the same result Query1.SQL property value.

```
Query1.MacroByName('StringMacro').Value := '''A string''';
Query1.MacroByName('StringMacro').AsString := 'A string';
```

Macros can be especially useful in scripts that perform similar operations on different objects. You can use macros that will be replaced with an object name. It allows you to have the same script text and to change only macro values. For example, the following is a script that creates a new user account and grants required privileges.

```
Script1.SQL.Add('CREATE USER &Username IDENTIFIED BY &Password;');
Script1.SQL.Add('GRANT &Privileges TO &Username;');
```

To execute the script for another user you do not have to change the script SQL property, you can just set required macro values.

You may also consider using macros to construct adaptable conditions in WHERE clauses of your statements.

See Also

- TMacro
- TCustomDADataSet.MacroByName
- TCustomDADataSet.Macros

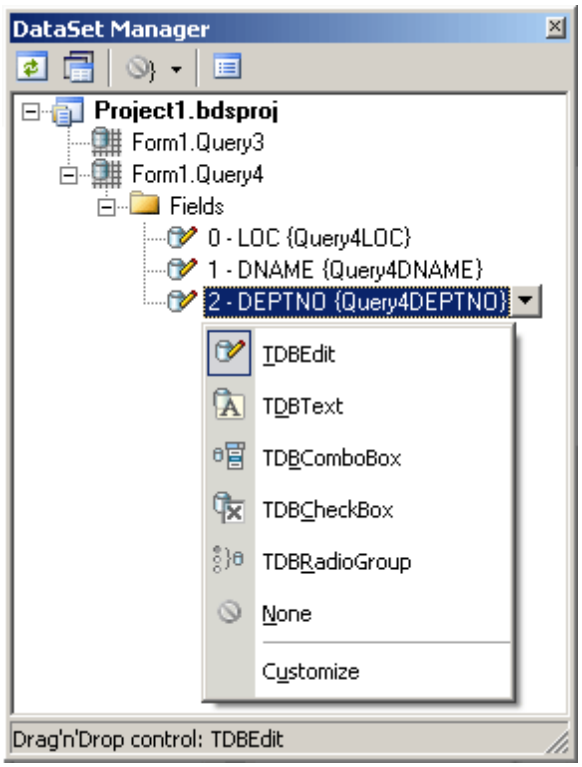
Request Support DAC Forum Provide Feedback

4.12 DataSet Manager

DataSet Manager window

The DataSet Manager window displays the datasets in your project. You can use the DataSet Manager window to create a user interface (consisting of data-bound controls) by dragging items from the window onto forms in your project. Each item has a drop-down control list where you can select the type of control to create prior to dragging it onto a form. You can

customize the control list with additional controls, including the controls you have created.



Using the DataSet Manager window, you can:

- Create forms that display data by dragging items from the DataSet Manager window onto forms.
- Customize the list of controls available for each data type in the DataSet Manager window.
- Choose which control should be created when dragging an item onto a form in your Windows application.
- Create and delete TField objects in the DataSets of your project.

Opening the DataSet Manager window

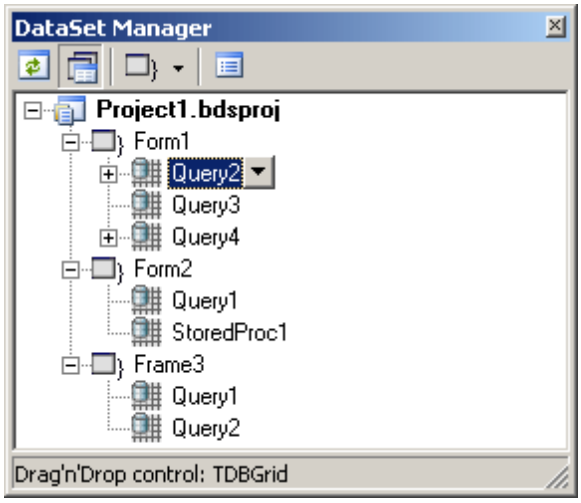
You can display the DataSet Manager window by clicking DataSet Manager on the Tools menu. You can also use IDE desktop saving/loading to save DataSet Manager window

position and restore it during the next IDE loads.

Observing project DataSets in the DataSet Manager Window

By default DataSet Manager shows DataSets of currently open forms. It can also extract DataSets from all forms in the project. To use this, click *Extract DataSets from all forms in project* button. This settings is remembered. Note, that using this mode can slow down opening of the large projects with plenty of forms and DataSets. Opening of such projects can be very slow in Delphi 6 and Borland Developer Studio 2006 and can take up to several tens of minutes.

DataSets can be grouped by form or connection. To change DataSet grouping click the *Grouping mode* button or click a down. You can also change grouping mode by selecting required mode from the DataSet Manager window popup menu.

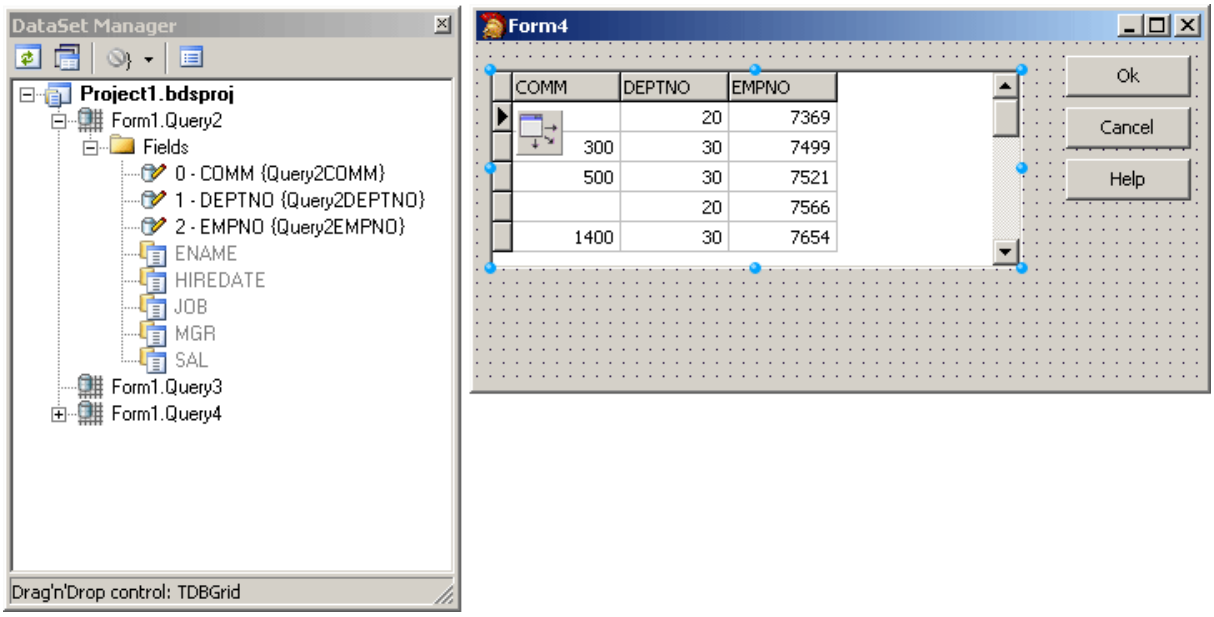


Creating Data-bound Controls

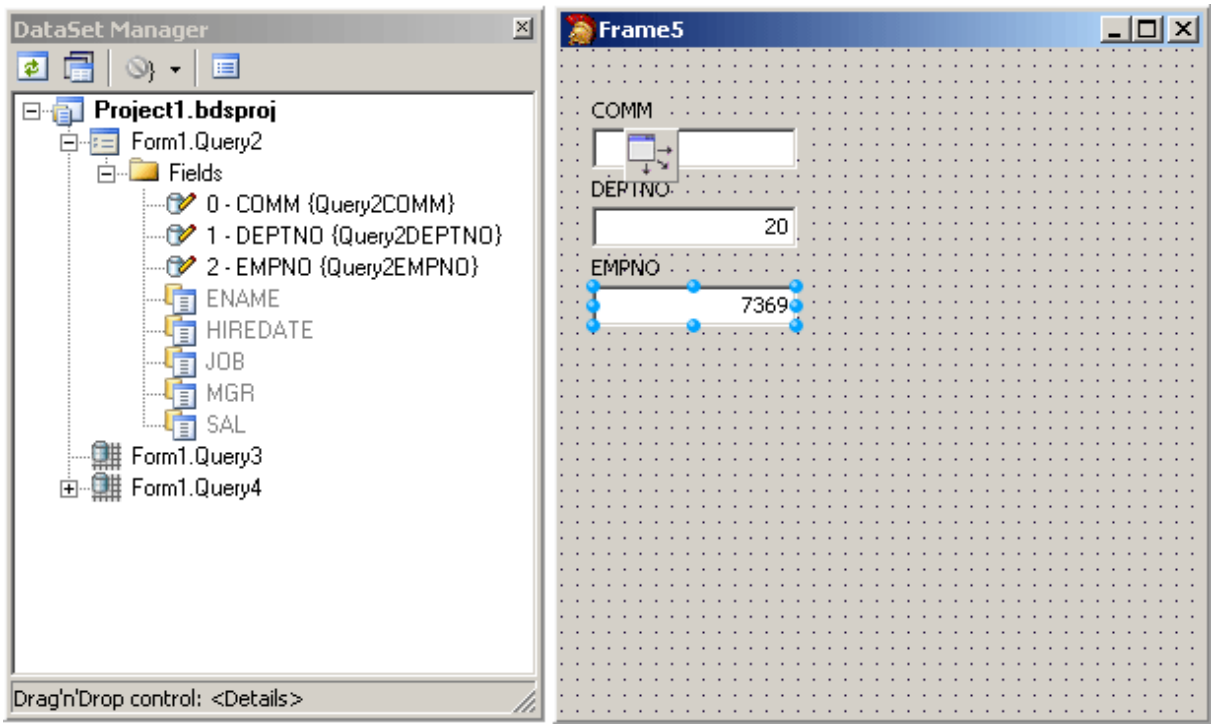
You can drag an item from the DataSet Manager window onto a form to create a new data-bound control. Each node in the DataSet Manager window allows you to choose the type of control that will be created when you drag it onto a form. You must choose between a Grid layout, where all columns or properties are displayed in a TDataGrid component, or a Details layout, where all columns or properties are displayed in individual controls.

To use grid layout drag the dataset node on the form. By default TDataSource and TDBGrid components are created. You can choose the control to be created prior to dragging by

selecting an item in the DataSet Manager window and choosing the control from the item's drop-down control list.

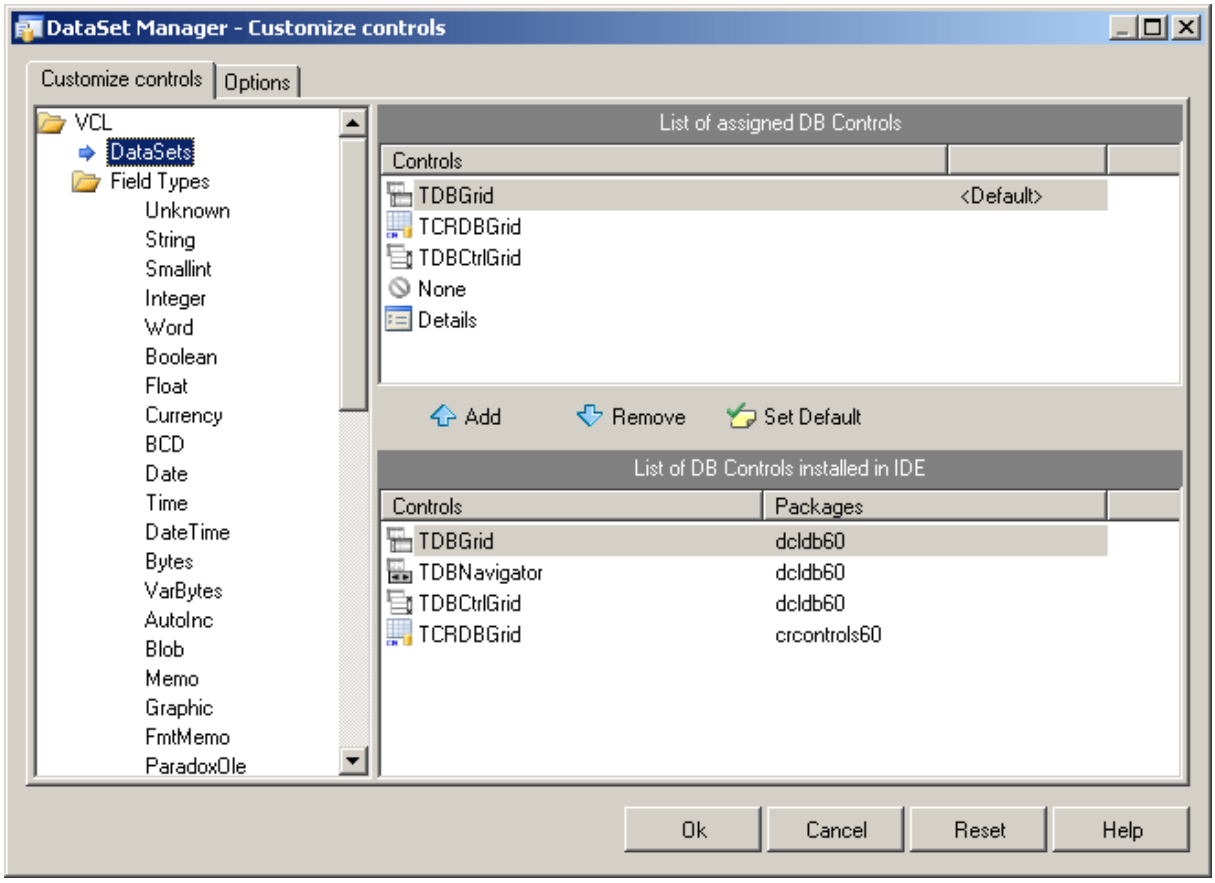


To use Details layout choose Details from the DataSet node drop-down control list in the DataSet Manager window. Then select required controls in the drop-down control list for each DataSet field. DataSet fields must be created. After setting required options you can drag the DataSet to the form from the DataSet wizard. DataSet Manager will create TDataSource component, and a component and a label for each field.



Adding custom controls to the DataSet Manager window

To add custom control to the list click the *Options* button on the DataSet Manager toolbar. A *DataSet Manager - Customize controls* dialog will appear. Using this dialog you can set controls for the DataSets and for the DataSet fields of different types. To do it, click DataSets node or the node of field of required type in *DB objects groups* box and use *Add* and *Remove* buttons to set required control list. You can also set default control by selecting it in the list of assigned DB controls and pressing *Default* button.



The default configuration can easily be restored by pressing Reset button in the *DataSet Manager - Options* dialog.

Working with TField objects

DataSet Manager allows you to create and remove TField objects. DataSet must be active to work with its fields in the DataSet Manager. You can add fields, based on the database table columns, create new fields, remove fields, use drag-n-drop to change fields order.

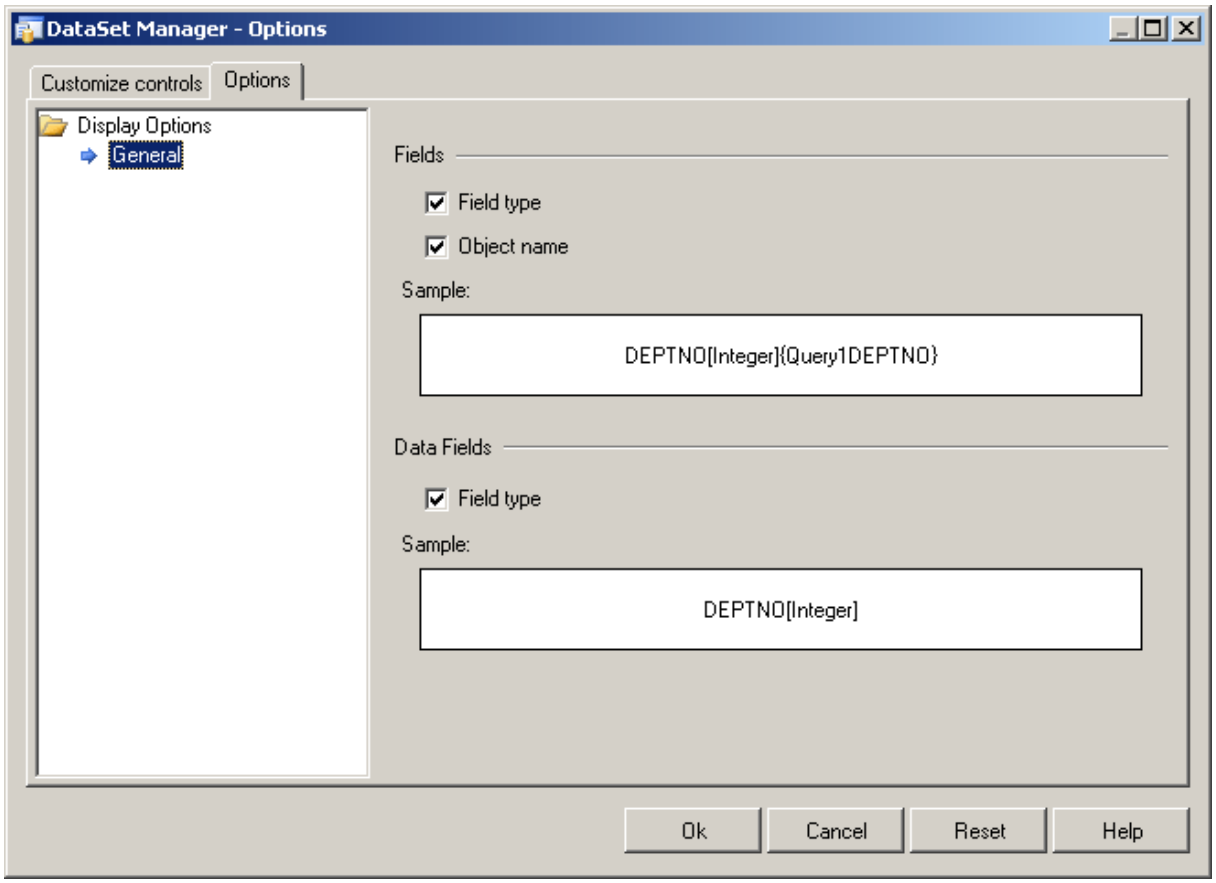
To create a field based on the database table column right-click the Fields node and select *Create Field* from the popup menu or press <Insert>. Note that after you add at least one field manually, DataSet fields corresponding to data fields will not be generated automatically when you drag the DataSet on the form, and you can not drag such fields on the form. To add all available fields right-click the Fields node and select *Add all fields* from the popup menu.

To create new field right-click the Fields node and select *New Field* from the popup menu or press <Ctrl+Insert>. The New Field dialog box will appear. Enter required values and press

OK button.

To delete fields select these fields in the DataSet Manager window and press <Delete>.

DataSet Manager allows you to change view of the fields displayed in the main window. Open the *Customize controls* dialog, and jump to the Options page.



You can chose what information will be added to names of the Field and Data Field objects in the main window of DataSet Manager. Below you can see the example.

Request Support DAC Forum Provide Feedback

4.13 Using Connection Pooling

Connection pooling enables an application to use a connection from a pool of connections that do not need to be reestablished for each use. Once a connection has been created and placed in a pool, an application can reuse that connection without performing the complete

connection process.

Using a pooled connection can result in significant performance gains, because applications can save the overhead involved in making a connection. This can be particularly significant for middle-tier applications that connect over a network or for applications that connect and disconnect repeatedly, such as Internet applications.

To use connection pooling set the Pooling property of the TCustomDAConnection component to True. Also you should set the PoolingOptions of the TCustomDAConnection. These options include MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime. Connections belong to the same pool if they have identical values for the following parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime, Server, Username, Password , TOraSession.Username, TOraSession.Server, TOraSession.ConnectMode, TOraSession.Options.UseOCI7 . When a connection component disconnects from the database the connection actually remains active and is placed into the pool. When this or another connection component connects to the database it takes a connection from the pool. Only when there are no connections in the pool, new connection is established.

ODAC connection pool is thread safe. Multiple threads of an application can connect to the database and disconnect from it using TCustomDAConnection components with pooling enabled at the same time. Connection pool uses the most optimal algorithms for fast and reliable work.

Connections in the pool are validated to make sure that a broken connection will not be returned for the TCustomDAConnection component when it connects to the database. The pool validates connection when it is placed to the pool (e. g. when the TCustomDAConnection component disconnects). If connection is broken it is not placed to the pool. Instead the pool frees this connection. Connections that are held in the pool are validated every 30 seconds. All broken connections are freed. If you set the PoolingOptions.Validate to True, a connection also will be validated when the TCustomDAConnection component connects and takes a connection from the pool. When some network problem occurs all connections to the database can be broken. Therefore the pool validates all connections before any of them will be used by a TCustomDAConnection component if a fatal error is detected on one connection.

The pool frees connections that are held in the pool during a long time. If no new connections are placed to the pool it becomes empty after approximately 4 minutes. This pool behaviour is intended to save resources when the count of connections in the pool exceeds the count that

is needed by application. If you set the PoolingOptions.MinPoolSize property to a non-zero value, this prevents the pool from freeing all pooled connections. When connection count in the pool decreases to MinPoolSize value, remaining connection will not be freed except if they are broken.

The PoolingOptions.MaxPoolSize property limits the count of connections that can be active at the same time. If maximum count of connections is active and some TCustomDAConnection component tries to connect, it will have to wait until any of TCustomDAConnection components disconnect. Maximum wait time is 30 seconds. If active connections' count does not decrease during 30 seconds, the TCustomDAConnection component will not connect and an exception will be raised.

You can limit the time of connection's existence by setting the PoolingOptions.ConnectionLifeTime property. When the TCustomDAConnection component disconnects, its internal connection will be freed instead of placing to the pool if this connection is active during the time longer than the value of the PoolingOptions.ConnectionLifeTime property. This property is designed to make load balancing work with the connection pool.

To force freeing of a connection when the TCustomDAConnection component disconnects, the RemoveFromPool method of TCustomDAConnection can be used. You can also free all connection in the pool by using the class procedures Clear or AsyncClear of TOraConnectionPoolManager. These procedures can be useful when you know that all connections will be broken for some reason.

It is recommended to use connection pooling with the DisconnectMode option of the TCustomDAConnection component set to True. In this case internal connections can be shared between TCustomDAConnection components. When some operation is performed on the TCustomDAConnection component (for example, an execution of SQL statement) this component will connect using pooled connection and after performing operation it will disconnect. When an operation is performed on another TCustomDAConnection component it can use the same connection from the pool.

Also, ODAC supports proxy connection pooling. When proxy pooling is used, TOraSession components can connect to a database with different Username and Password properties but all connections in the pool use PoolingOptions.ProxyUsername and PoolingOptions.ProxyPassword. When connecting, TOraSession component creates a new connection. It uses Username, Password properties and a connection from the pool as a

proxy connection. The proxy connection pool allows you to use a single pool for all sessions with different Username and Password properties.

You can use OCI connection pooling or MTS connection pooling. To use these types of pooling set the PoolingOptions.PoolType to ptOCI or ptMTS. In this case the pool is created and managed by the Oracle client or by MTS.

See Also

- TCustomDAConnection.Pooling
- TCustomDAConnection.PoolingOptions
- TOraSession.PoolingOptions
- Working with Disconnected Mode

Request Support DAC Forum Provide Feedback

4.14 Automatic Key Field Value Generation

When editing dataset it is often convenient to generate key field(s) values automatically instead of filling them manually. In the most common way application developer generates primary key value basing it on previously created sequence. There are three ways to do it.

First, application independent way - developer creates AFTER INSERT trigger that fills the field value. But here he faces the problem with getting value inserted by trigger back to dataset. This problem can be easily solved in ODAC by specifying return parameters. For instance:

```
...
// suppose that AFTER INSERT trigger fills DepNo field
OraQuery.SQL.Text := 'SELECT DepNo, DepName, Location FROM Department';
OraQuery.SQLInsert.Text := 'INSERT INTO Department (DepNo, DepName, Location
                           'VALUES(DepNo, DepName, Location) ' +
                           'RETURNING DepNo INTO :DepNo';
...
```

Second way is custom key field value generation. Developer can fill key field value in TOraDataSet.BeforePost event handler. But in this case he should manually execute query and retrieve sequence value. So this way may be useful only if some special value processing is needed.

The third way, using KeySequence is the simplest. Developer only needs to specify two properties and key field values are generated automatically. There is no need to create trigger or perform custom BeforePost processing.

```
...
OraQuery.SQL.Text := 'SELECT DepNo, DepName, Location FROM Department';
OraQuery.KeyFields := 'DepNo';          // key field
OraQuery.KeySequence := 'DepSequence'; // sequence that will generate values
...
```

See also

- KeySequence

Request Support DAC Forum Provide Feedback

4.15 TOraLoader Component

There are ceses when you need to put large amount of data to Oracle database. Of course, you may construct INSERT SQL statement and execute it with TOraSQL component. But it takes a lot of time. You can greatly speed up loading time of data by using DML array features. Oracle 8i has better way to do it. With Oracle 8i using the direct path load interface is possible. The direct path load interface allows to access the direct path load engine of the Oracle database server to perform the functions of the Oracle SQL*Loader utility. This functionality provides the ability to load data from external files into Oracle database objects, either a table or a partition of a partitioned table.

ODAC simplifies using direct path load interface by TOraLoader component. TOraLoader allows you to load various formatted data. The capability of TOraLoader component to load various formatted data is reached by reading external data in writing method itself.

To write your own loader you should:

- create TOraLoader component;
- set name of loading table to TableName;
- create and customize columns which will be loaded (use TOraLoader component editor at design time);
- write your own event handler: OnGetColumnData or OnPutData
- call Load method to start loading.

See Also

- TOraLoader
- TDPColumn

Request Support DAC Forum Provide Feedback

4.16 TOraTransaction Component

TOraTransaction component can be used to manage either local or distributed (global) transactions.

To start local transaction with TOraTransaction component, set DefaultSession property of the component to a session on which transaction will be performed. Set IsolationLevel property optionally. Then call StartTransaction method of the TOraTransaction component. To manage transaction use Commit, Rollback, Savepoint, RollbackToSavepoint methods.

Global transactions can be performed on one or more sessions connected to the same or to the different databases. These sessions can be established from different applications and computers. On each of these sessions a separate branch of transaction is performed. Global transaction can be coordinated either by internal mechanism of TOraTransaction or Microsoft Transaction Manager DTC. This behavior can be tuned by GlobalCoordinator property. In case of using internal mechanism you should specify TransactionId and BranchQualifier for each session to identify global transaction. Global transaction can be managed using Commit, Rollback, Savepoint, RollbackToSavepoint, Detach and Resume methods of TOraTransaction. If an OraSession uses global transaction, it must have non-empty InternalName property. For more information about global transaction please refer to Oracle documentation.

Note: transaction will be global if either TransactionId or TransactionName property is set or if GlobalCoordinator property is gcMTS.

Here is a sample code that starts and commits global transaction:

```
var
Id: TBytes;
begin
  OraSession1.InternalName := 'SampleName1';
  OraSession2.InternalName := 'SampleName2';
  OraSession1.Connect;
```

```
  OraSession2.Connect;
  SetLength(Id, 2);
  id[0] := 7; id[1] := 3;
  OraTransaction.TransactionId := Id;
  SetLength(Id, 1);
  id[0] := 1;
  OraTransaction.AddSession(OraSession1, Id);
  id[0] := 2;
  OraTransaction.AddSession(OraSession2, Id);
  OraTransaction.StartTransaction;
  OraSQL1.Session := OraSession1;
  OraSQL2.Session := OraSession2;
  OraSQL1.Execute;
  OraSQL2.Execute;
  OraTransaction.Commit;
end;
```

The following example demonstrates a global transaction with two branches created from different applications. After performing update these applications detach their transaction branches. Then third application (transaction manager) resumes all branches of the transaction and performs two-phase commit.

```
// Appication 1
var
  Id: TBytes;
begin
  OraSession.InternalName := 'SampleName1';
  OraSession.Connect;
  SetLength(Id, 2);
  id[0] := 7; id[1] := 3;
  OraTransaction.TransactionId := Id;
  SetLength(Id, 1);
  id[0] := 1;
  OraTransaction.AddSession(OraSession, Id);
  OraTransaction.StartTransaction;
  OraSQL.Execute;
  OraTransaction.Detach;
end;
// Appication 2
var
  Id: TBytes;
begin
  OraSession.InternalName := 'SampleName2';
  OraSession.Connect;
  SetLength(Id, 2);
  id[0] := 7; id[1] := 3;
  OraTransaction.TransactionId := Id;
  SetLength(Id, 1);
  id[0] := 2;
  OraTransaction.AddSession(OraSession, Id);
  OraTransaction.StartTransaction;
  OraSQL.Execute;
  OraTransaction.Detach;
end;
// Application 3 (transaction manager that commits transaction)
```

```
var
  Id: TBytes;
begin
  OraSession1.Connect;
  OraSession2.Connect;
  SetLength(Id, 2);
  id[0] := 7; id[1] := 3;
  OraTransaction.TransactionId := Id;
  SetLength(Id, 1);
  id[0] := 1;
  OraTransaction.AddSession(OraSession1, Id);
  id[0] := 2;
  OraTransaction.AddSession(OraSession2, Id);
  OraTransaction.Resume;
  OraTransaction.Commit;
end;
```

The next example demonstrates using distributed transaction coordinated by MTS DTC:

```
begin
  OraSession1.InternalName := 'SampleName1';
  OraSession2.InternalName := 'SampleName2';
  OraSession1.Connect;
  OraSession2.Connect;
  OraTransaction.GlobalCoordinator := gcMTS;
  OraTransaction.AddSession(OraSession1);
  OraTransaction.AddSession(OraSession2);
  OraTransaction.StartTransaction;
  OraSQL1.Session := OraSession1;
  OraSQL2.Session := OraSession2;
  OraSQL1.Execute;
  OraSQL2.Execute;
  OraTransaction.Commit;
end;
```


Request Support DAC Forum Provide Feedback

4.17 TOraQueue, TOraQueueAdmin and TOraQueueTable Components

TOraQueue, TOraQueueAdmin and TOraQueueTable components provide access to Oracle Streams Advanced Queuing. Oracle Streams AQ provides database-integrated message queuing functionality. It is built on top of Oracle Streams and leverages the functions of Oracle Database so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services and HTTP(S).

Because Oracle Streams AQ is implemented in database tables, all operational benefits of high availability, scalability, and reliability are also applicable to queue data. Standard database features such as recovery, restart, and security are supported by Oracle Streams

AQ. Like other database tables, queue tables can be imported and exported.

When applications communicate with each other, producer applications enqueue messages and consumer applications dequeue messages. At the basic level of queuing, one producer enqueues one or more messages into one queue. Each message is dequeued and processed once by one of the consumers. A message stays in the queue until a consumer dequeues it or the message expires. A producer may stipulate the delay before the message is available for the consumption, and the time after which the message expires. Likewise, a consumer may wait when trying to dequeue a message if no message is available. An agent program or application may act both as a producer and a consumer.

TOraQueue component provides access to functionality of DBMS_AQ Oracle package. User must have AQ_USER_ROLE to work with this component. TOraQueue component can be used to enqueue and dequeue messages from the given queue. Queue message includes a payload and message properties. Type of payload is defined for each queue. This can be Oracle object type or 'RAW' type. RAW payload contains any array of bytes. To use TOraQueue component set its Session and QueueName properties. Then one of Enqueue method overloads can be used to enqueue message.

To enqueue message with RAW payload use the Enqueue method overload with string or TBytes Payload parameter. For example:

```
var
  MsgProp: TQueueMessageProperties;
begin
  MsgProp := TQueueMessageProperties.Create;
  try
    MsgProp.Priority := 2;
    MsgProp.Delay := 30; // delay in sec before message can be dequeued
    MsgProp.Expiration := 180; // message expiration in sec
    OraQueue.Enqueue('123', MsgProp);
  finally
    MsgProp.Free;
  end;
end;
```

To enqueue message to queue with object payload use Enqueue method overload with TOraObject Payload parameter. For example:

```
var
  MsgProp: TQueueMessageProperties;
  Payload: TOraObject;
begin
  MsgProp := TQueueMessageProperties.Create;
  try
    MsgProp.Priority := 2;
```

```
    MsgProp.Delay := 30; // delay in sec before message can be dequeued
    MsgProp.Expiration := 180; // message expiration in sec
    Payload := TOraObject.Create;
    try
      Payload.AllocObject(OraSession.OCISvcCtx, 'OBJ_MES');
      Payload.AttrAsInteger['A'] := 3;
      Payload.AttrAsString['B'] := 'Hello';
      OraQueue.Enqueue(Payload, MsgProp);
    finally
      Payload.Free;
    end;
  finally
    MsgProp.Free;
  end;
end;
```

To dequeue message, use one of TOraQueue.Dequeue method overloads. If there are no messages available for dequeuing, Dequeue method will wait until a message will be available. The following example demonstrates the usage of Dequeue method for a queue with object payload:

```
var
  Payload: TOraObject;
  a: integer;
  b: string;
begin
  Payload := TOraObject.Create;
  try
    OraQueue.Dequeue(Payload);
    a := Payload.AttrAsInteger['A'];
    b := Payload.AttrAsString['B'];
  finally
    Payload.Free;
  end;
end;
```

To get notification when a message available to dequeuing appears in the queue use TOraQueue.Listen method or set AsyncNotification property to True and write OnMessage event handler.

Listen method listens on one or more queues on behalf of a list of agents. This method waits until a message is available in one of the queues and then returns the agent that corresponds this queue. Listen method should be called in separate thread to avoid program blocking. For example:

```
procedure TListenThread.Execute;
var
  Agents: TQueueAgents;
  Agent: TQueueAgent;
  Message: string;
begin
  Agents := TQueueAgents.Create;
```

```
  try
    Agents.Add.Address := 'QUEUE1'; // Address property should contain the q
    Agents.Add.Address := 'QUEUE2';
    Agent := TQueueAgent.Create;
    try
      while not Terminated do begin
        OraQueue.Listen(Agents, Agent);
        OraQueue.Dequeue(Message);
      end;
    finally
      Agent.Free;
    end;
  finally
    Agents.Free;
  end;
end;
```

TOraQueueAdmin and TOraQueueTable components are used to administrate queues. User must have AQ_ADMINISTRATOR_ROLE to work with these components. TOraQueueAdmin and TOraQueueTable components can be used to create and drop queues and queue tables, alter queue properties.

Queue can be persistent or non-persistent. To create persistent queue first a queue table must be created. Use TOraQueueTable component to create a queue table. Set properties of the component to required values and then call CreateQueueTable method. Use AlterQueueTable method to alter a queue table properties and DropQueueTable method to drop a queue table.

When a queue table is created, TOraQueueAdmin component can be used to create a queue. Set properties of the component to required values and then call CreateQueue method. Use AlterQueue method of TOraQueueAdmin to alter a queue properties and DropQueue method to drop a queue.

Before messages can be enqueued and dequeued enqueuing and dequeuing must be started on the queue. Use StartQueue method of TOraQueueAdmin to start enqueuing and dequeuing on the queue.

Request Support DAC Forum Provide Feedback

4.18 TOraChangeNotification Component

TOraChangeNotification component is used to register queries with the database and receive notifications in response to DML or DDL changes on the objects associated with the queries. The notifications are published by the database when the DML or DDL transaction commits.

When the database issues change notification, it can contain some or all of the following information:

- Names of the modified objects. For example, the notification can specify that the hr.employees table was changed.

- The type of change. For example, the message specifies whether the change was caused by an INSERT, UPDATE, DELETE, ALTER TABLE, or DROP TABLE.

- The ROWIDs of the changed rows and the type of DML that changed them.

- Global events such as STARTUP and SHUTDOWN (consistent only). In a Real Applications Cluster, the database delivers a notification when the first instance on the database starts or the last instance shuts down.

The notification contains only metadata about the changed rows or objects rather than the changed data itself. For example, the database does not notify the client that monthly salary increased from 5000 to 6000. To obtain more recent values for the changed objects or rows, the client must query the database based on the information contained in the notification.

Database Change Notification is useful for an application that caches query result sets on mostly read-only objects in the mid-tier to avoid network round trips to the database. Such application can create a registration on the queries it is interested in caching using the change notification service. On changes to objects referenced inside those queries, the database publishes a change notification when the underlying transaction commits. In response to the notification, the application can refresh its cache by re-executing the queries.

TOraChangeNotification component represents dependency between an application and an Oracle database based on the database events in which the application is interested. To create subscription place on the form TOraChangeNotification component and assign it to ChangeNotification properties of datasets. When you open a dataset subscription on changes to database tables that dataset selects data from will be created. All datasets that use one TOraChangeNotification component share one change notification subscription. A dataset is registered in subscription when you open this dataset.

Components derived from TOraDataSet can automatically refresh their data in response to change notification. To enable autorefresh set ReflectChangeNotify in TOraDataSet.Options to True. Dataset refreshes when data in one of the tables that appear in query FROM clause is changed. If dataset's SQL has ROWID in SELECT clause and changed table corresponds

UpdatingTable property of TOraDataSet then only changed rows are refreshed with RefreshRecord method. If TOraDataSet.SQLRefresh property is set it must use ROWID. Otherwise refreshing may be incorrect. If TOraDataSet.SQL property doesn't contain ROWID in SELECT clause or data is changed not in UpdatingTable but in other table that appears in query FROM clause the dataset performs full refresh with Refresh method.

Change notification can be handled manually using OnChange event of TOraChangeNotification component. For example:

```
procedure TForm1.OraChangeNotificationChange(Sender: TObject;
  NotifyType: TChangeNotifyEventType; TableChanges: TNotifyTableChanges);
var
  i, j: integer;
begin
  if NotifyType <> cneObjChange then
    Exit;
  for i := 0 to TableChanges.Count - 1 do begin
    Memo.Lines.Add(TableChanges[i].TableName);
    if cnoAllRows in TableChanges[i].Operations then
      Continue;
    for j := 0 to TableChanges[i].RowChanges.Count - 1 do
      Memo.Lines.Add(TableChanges[i].RowChanges[j].RowId);
  end;
end;
```

For the best performance of change notification, the following guidelines are presented. Registered objects are few and mostly read-only and modifications to these objects are rather an exception than a rule. If the object is extremely volatile, then it will cause a large number of invalidation notifications to be sent, and potentially a lot of storage in the invalidation queue on the server. If there are frequent and a large numbers of notifications, the OLTP throughput can be slowed down due to the overhead of the notifications generation.It is also a good idea to keep the number of duplicate registrations on any given object low (ideally one) in order to avoid the same notification message being replicated to multiple recipients.

See Also

- TOraDataSet.Options
- TOraChangeNotification

4.19 Working With Data

This section describes the basics of working with data. It contains the information on how to work with BLOB, CLOB, XMLTYPE, VARRAY data types, Unicode Character Data, Objects, PL/SQL Tables, etc.

- BLOB and CLOB Data Types
- Unicode Character Data
- Objects
- XMLTYPE Data Type
- VARRAY Data Type
- DML Array
- PL/SQL Tables

Request Support DAC Forum Provide Feedback

4.19.1 BLOB and CLOB Data Types

ODAC components support Oracle 8 BLOB and CLOB data types. You can retrieve values of LOB fields using TOraQuery component the same way as you do for LONG or LONG RAW fields. The difference with usage of LOB data type becomes evident when you need to access these fields in SQL DML and PL/SQL statements.

For BLOB and CLOB data types only LOB locators (pointers to data) are stored in table columns; actual BLOB and CLOB data is stored in separate tablespace. This is the difference to the way that data of LONG or LONG RAW types is stored in database – tables hold their immediate values.

When accessing LOB column, it is the locator which is returned, not the value itself as in the case with LONG or LONG RAW data types.

For example consider this table definition.

```
CREATE TABLE ClobTable (
  Id NUMBER,
  Name VARCHAR2(30),
  Desc CLOB
)
```

To update Desc column of this table we need to create CLOB, get its locator, write CLOB data using this locator and excecute UPDATE statement to write the locator into the table field.

The following SQL statement can be used to update ClobTable:

```
UPDATE ClobTable
SET
  Name = :Name,
  Desc = EMPTY_CLOB()
WHERE
  Id = :Id
RETURNING
  Desc
INTO
  :Desc
```

You can use EMPTY_BLOB or EMPTY_CLOB Oracle function to create empty LOB. After executing this statement LOB locator is returned into Desc parameter. Then ODAC writes LOB data into database using this locator. You must set ParamType of Desc parameter to ptInput.

It is important for ODAC to use ParamType property of parameters in LOB operations. If ParamType is ptInput ODAC writes data to a server, if ParamType is ptOutput it reads data.

Another way to update ClobTable is using temporary LOB. Set TemporaryLobUpdate in TOraDataSet.Options to True. Then ODAC will create temporary LOB and write data to it before executing SQL statement.

When TemporaryLobUpdate option is True following statement can be used to update ClobTable:

```
UPDATE ClobTable
SET
  Name = :Name,
  Desc = :Desc
WHERE
  Id = :Id
```

ODAC will initialize Desc parameter with locator of temporary LOB before executing this statement.

TemporaryLobUpdate option should be set to True when calling stored procedure with IN or IN OUT LOB parameter. To call procedure:

```
CREATE OR REPLACE
PROCEDURE ClobTableUpdate (p_Id IN NUMBER, p_Name IN VARCHAR2,
                           p_Desc IN CLOB)
```

```
IS
BEGIN0
  UPDATE ClobTable
  SET
    Name = p_Name,
    Desc = p_Desc
  WHERE
    Id = p_Id;
END;
```

the following code can be used:

```
OraStroredProc.Options.TemporaryLobUpdate := True;
OraStroredProc.StoredProcName := 'ClobTableUpdate';
OraStroredProc.Prepare;
OraStroredProc.ParamByName('p_Id').AsInteger := Id;
OraStroredProc.ParamByName('p_Name').AsString := Name;
OraStroredProc.ParamByName('Desc').ParamType := ptInput;
OraStroredProc.ParamByName('Desc').AsOraClob.LoadFromFile(FileName);
OraStroredProc.Execute;
```

Note that LOB parameter can have ptInputOutput type only when TemporaryLobUpdate option is set toTrue. Otherwise the type of LOB parameter must be ptInput or ptOutput.

You can also use dtBlob and dtMemo datatypes with LOB parameters to write ordinary DML statements. In this case Oracle automatically converts LONG and LONG ROW values to CLOB or BLOB data.

It is possible to control the way LOB objects are handled while the application fetches records from the database. LOBs can be fetched either with other fields to the application or on demand.This is determined by DeferredLobRead option in TOraDataSet component. Setting TOraDataSet.Options.DeferredLobRead to false allows to reduce traffic over the network since LOBs are only transferred on demand and to use less memory on the client side because returned record sets do not hold contents of LOB fields.

For managing LOB compression, use TCustomDADataSet.Options.CompressBlobMode. LOBs can be stored compressed on the client side, on the server side (in database) or on both sides. By default it has cbNone value, that means no compression is provided. Use cbClient value to store compressed LOBs on client side. This saves client memory. LOB data is stored unchanged in a database, other applications can read these LOBs as usual. If cbServer value is used, LOB data is stored compressed in the database. It's decompressed on the client side. This saves server disk space and network traffic. Other application can't process compressed LOB data as usual. To use compressed LOB data both on the client and server sides use cbClientServer value. To use cbClient, cbServer, cbClientServer and cbNone constants you should add the MemData unit to the uses clause.

Set TOraDataSet.Options.CacheLobs to False to access streamed LOB values on the server side without caching LOBs on the client side. Only requested portions of data are fetched in that case. Setting CacheBlobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since only required data are fetched, less memory is needed on the client side because LOB data is not cached on client side. This option doesn't make sense if DeferredLobRead is set to False because in that case all LOB values are fetched to the dataset.

Note: Internal compression functions are available in Borland Developer Studio 2006, Delphi 2005 and Delphi 7. To use BLOB compression under Delphi 6 and C++Builder you should use your own compression functions. To use them set CompressProc and UncompressProc variables declared in MemUtils unit.

```
type
  TCompressProc = function(dest: IntPtr; destLen: IntPtr;
  const source: IntPtr; sourceLen: longint): longint;
  TUncompressProc = function(dest: IntPtr; destlen: IntPtr;
  source: IntPtr; sourceLne: longint): longint;
var
  CompressProc: TCompressProc;
  UncompressProc: TUncompressProc;
```

You can compress and decompress a single LOB. To do it set the TOraLob.Compressed property. Set it to True to compress LOB data and to False to decompress LOB data.

Note that using compression and decompression operations will raise CPU usage and can reduce application performance.

See Also

- TOraLob
- TDAParam.ParamType
- TCustomDADataSet.Options
- TOraDataSet.Options
- BlobPic demo
- Clob demo

Request Support DAC Forum Provide Feedback

4.19.2 Unicode Character Data

Symbolic information in Oracle can be retrieved for the user as a different character encoding according to the query. Oracle supports a number of encoding formats including Unicode. ODAC components support UTF-16 Unicode encoding formats for data fields with OCI 8.0 or higher. Any character of any language can be represented in UTF-16.

ODAC allows to represent string data using string and WideString types. You can use TOraSession.UseUnicode property to enable this behaviour.This property value affects the description of queries and stored procedures. TOraSession.UseUnicode property does not influence the parameters' types that were set manually.

Suppose that SIMPLE_TYPES table is created as:

```
CREATE TABLE SIMPLE_TYPES (
  ID NUMBER(6) NOT NULL,
  F_CHAR CHAR(250),
  F_VARCHAR VARCHAR2(300),
  F_RAW RAW(250),
)
```

Suppose we open following SELECT statement in dataset

```
SELECT a.RowId, a.* FROM SIMPLE_TYPES a
```

If TOraSession.UseUnicode is set to False you get the next fields list after dataset open:

```
RowId:     TStringField
ID:        TIntegerField
F_CHAR:    TStringField
F_VARCHAR: TStringField
F_RAW:     TVarBytesField
```

When you set TOraSession.UseUnicode to True the string fields' type changes:

```
RowId:     TWideStringField
ID:        TIntegerField
F_CHAR:    TWideStringField
F_VARCHAR: TWideStringField
F_RAW:     TWideStringField
```

Fields of TWideStringField type hold rows in UTF-16 Unicode format. To get the value of the fields you can use TWideStringField.Value property. You can use FlatBuffers, LongString, FieldsAsString, RawAsString, TrimFixedChar options of TOraDataSet which are compatible with TOraSession.UseUnicode.

To use Unicode values as parameters previously you need to set the value of data type field to ftWideString or ftFixedWideChar for the fields of VARHAR or CHAR types accordingly. Otherwise after the execution of AsWideString or AsString operation data type field will be ftString by default.

```
var
  WS: WideString;
begin
...
  with OraQuery1 do begin
    Close;
    SQL.Text:=
      'SELECT * from SIMPLE_TYPES '+
      'WHERE '+
      '  F_CHAR = :F_CHAR';
    Params[0].DataType := ftFixedWideChar;
    Params[0].AsWideString := WS;
    Open;
...
```

If parameter has Unicode data type value, assigning value by using AsString property converts String to WideString. And vice versa, if parameter doesn't have Unicode data type value, assigning value by AsWideString property converts WideString into String.

Also Unicode encoding is supported for ROWID, NUMBER, INTERVAL, TIMESTAMP, RAW, CLOB.

CLOB data type supports string data in UTF-16 Unicode encoding. You can set TOraSession.UseUnicode property to True and get TMemoField of ftOraClob blob type. You can update CLOB field and set its value to Unicode string the following way:

```
var
  WS: WideString;
begin
...
```

```
  with OraQuery1 do begin
    SQL.Text:=
      'UPDATE ODAC_CLOB '+
      'SET '+
      '  Value = EMPTY_CLOB() '+
      'WHERE '+
      '    ID = :ID '+
      'RETURNING '+
      '  Value '+
      'INTO '+
      '  :Value ';
    ParamByName('ID').AsFloat:=1;
    ParamByName('Value').ParamType := ptInput;
    ParamByName('Value').AsCLOBLocator.IsUnicode := True;
    ParamByName('Value').AsCLOBLocator.Write(0, Length(WS)*2, PWideChar(WS))
    // or
    ParamByName('Value').AsWideString := WS;
    Execute;
...
```

See Also

- TOraSession.Options
- TOraDataSet.Options
- TOraDataSet.OptionsDS

Request Support DAC Forum Provide Feedback

4.19.3 Objects

ODAC allows you to query and update columns of Oracle object type. You can access to attributes of object column as to fields of dataset using TDataSet.FieldByName or TDataSet.Fields. Dataset represents object attributes in two ways depending on the value of ObjectView property. If ObjectView is True attributes are stored hierarchically in the Fields property, that means any attributes of the object column are represented by child field of the object field and don't appear sequentially after the object field in the TFields.Fields array. When ObjectView is False, the attributes are stored sequentially in the Fields property, that means any child fields of the object field are stored after the object field as siblings in the Fields array.

For example we have such types and table

```
CREATE TYPE TAddress AS OBJECT (
  Country VARCHAR2(30),
  City VARCHAR2(30),
  Street VARCHAR2(30),
```

```
  Apartment NUMBER
);
CREATE TYPE TPerson AS OBJECT (
  Name VARCHAR2(30),
  Address TAddress,
  Phone VARCHAR2(20),
  BirthDate DATE
);
CREATE TABLE ODAC_Emp (
  Person TPerson,
  Job VARCHAR2(9),
  HireDate DATE,
  Sal NUMBER(7,2)
);
```

If you execute this query

```
SELECT * FROM ODAC_Emp
```

to learn the name of an employee you can write

```
Value:= Query.FieldByName('PERSON.NAME').AsString;
```

If ObjectView is True object column is represented by TADTField and to access the object attribute use child field by TADTField.Fields

```
Value:= TADTField(Query.FieldByName('PERSON')).
        Fields.FieldByName('NAME')).AsString;
```

Another way to get the value of an attribute is using TOraObject. Use TOraDataSet.GetObject method to get reference to the needed object.

So, the previous example may be rewritten this way

```
Value := Query.GetObject('PERSON').AttrAsString['NAME'];
```

Also ODAC supports object parameters. Use this feature when writing statements to update dataset rows. So, to insert a new row in ODAC_Emp table it is enough to assign this statement to SQLInsert property.

```
INSERT INTO ODAC_Emp
  (PERSON, JOB, HIREDATE, SAL)
VALUES
  (:PERSON, :JOB, : HIREDATE, :SAL)
```

To execute this INSERT statement only by TOraQuery or TOraSQL component use TOraParam.AsObject property to set attributes' value of PERSON parameter. But before you should assign dtObject to TOraParam.DataType property and allocate object handle by TOraParam.AllocObject method.

```
var
  OraSQL: TOraSQL;
. . .
```

```
  OraSQL.SQL.Text := 'INSERT INTO ODAC_Emp' +
                     '(PERSON, JOB, HIREDATE, SAL)' +
                     'VALUES (:PERSON, :JOB, : HIREDATE, :SAL)';
  with OraSQL.ParamByName('Person').AsObject do begin
    AllocObject(OraSession.OCISvcCtx, 'TPerson');
    AttrAsString['Name'] := 'JON';
    AttrAsString['Address.Country'] := 'USA';
    AttrAsString['Address.City'] := 'Boston';
    AttrAsInteger['Address.Apartment'] := 133;
    AttrAsDateTime['BirthDate'] := EncodeDate(1970, 7, 23);
  end;
  OraSQL.ParamByName('Job').AsString := 'MANAGER';
  OraSQL.ParamByName('HireDate').AsDateTime := EncodeDate(1998, 5, 14);
  OraSQL.ParamByName('Sal').AsInteger := 1700;
  OraSQL.Execute;
```

See Also

- TOraObject
- Object demo

Request Support DAC Forum Provide Feedback

4.19.4 XMLTYPE Data Type

Oracle 9i introduced a new data type, XMLTYPE, to facilitate native handling of XML data in the database. XMLTYPE has built-in member functions that operate on XML content. For example, you can use XMLTYPE functions to create, extract, and index XML datastored in Oracle 9i database. XMLTYPE data type can be used as the data type of columns in tables and views. XMLTYPE data type uses following storage methods:

- Large objects (LOBs). LOB storage maintains content accuracy to the original XML (white spaces and all). Here the XML documents are stored composed as whole documents like files. XML stored as a whole document in the database and retrieve it as a whole document. You do not need to perform piece-wise updates on XML documents.
- Structured storage (tables and views). Structured storage maintains DOM (Document Object Model) fidelity.

ODAC can work with fields of XMLTYPE data type. For example,suppose we have following table with XMLTYPE field:

```
CREATE TABLE xml_tab(
  ID NUMBER(10),
  XMLField XMLTYPE
```

```
)
```

After creating TOraQuery and execution of the next SELECT statement

```
SELECT * FROM xml_tab
```

TOraXMLField object will be created for XMLTYPE type which holds retrieved XML document. XMLTYPE fields are cached in ODAC on opening a table. Update of XMLTYPE fields will be posted to the server after the execution of corresponding DML query.

You can update or append XMLTYPE fields to the table the following way:

```
Edit; // or Insert, Append;
TOraXMLField(FieldByName('X')).AsXML.AsString :=
  '<root> <node1>v1</node1> <node2 name1=''222''>v2</node2> <node1>v3</node1
Post;
```

Or you can write:

```
Edit; // or Insert, Append;
TOraXMLField(FieldByName('X')).AsString :=
  '<root> <node1>v1</node1> <node2 name1=''222''>v2</node2> <node1>v3</node1
Post;
```

You can use TOraXMLField.AsXML property to get XMLTYPE document as an TOraXML object. Using this object you can query XMLTYPE data and extract its portions by calling TOraXML.Exists and TOraXML.Extract functions. Both these functions use a subset of the W3C XPath recommendation to navigate through the document. XMLTYPE uses the built-in Oracle XML parser and processor, and that's why it provides better performance and scalability when used inside the server. TOraXML.Transform function takes in XMLTYPE instance and XSLT stylesheet. It applies the stylesheet to the XML document and returns a transformed XML instance.

You can treat the XMLTYPE as a parameter in DML statements as shown below.

```
Close;
SQL.Text := 'UPDATE xml_tab SET  XMLField = :XMLField WHERE ID = 101';
with Params[0].AsXML do begin
  OCISvcCtx := OraSession1.OCISvcCtx;
  AsString := '<test></test>';
end;
Execute;
```

Or you can create the XMLTYPE value from TOraLob.

```
with Params[0].AsXML do begin
  OraLob := TOraLob.Create(OraSession1.OCISvcCtx);
  try
    OraLob.CreateTemporary(ltClob);
    OraLob.Write(0, Length('<test_lob></test_lob>'), PChar('<test_lob></test
    OraLob.WriteLob;
```

```
    AllocObject(OraSession1.OCISvcCtx, OraLob);
  finally
    OraLob.Free;
  end;
end;
Execute;
```

XML Schema is a schema definition language written in XML. It can be used to describe the structure and other various semantics of conforming instance documents. When using Oracle XML DB, you must first register your XML schema. Than you can use the XML schema URLs while creating XMLTYPE tables, columns, and views. You can use XML schema to declare which elements and attributes can be used and what kinds of element nesting, and data types are allowed in the XML documents that are being stored or processed. For example, user can create XML document based on the following schema, create table with XMLTYPE field and initialize the field value.

```
declare
doc varchar2(1000) :=
'<schema targetNamespace="http://www.oracle.com/PO.xsd"
  xmlns:po="http://www.oracle.com/PO.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="PurchaseOrderType">
    <sequence>
      <element name="PONum" type="decimal"/>
      <element name="Company">
        <simpleType>
          <restriction base="string">
            <maxLength value="100"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
  <element name="PurchaseOrder" type="po:PurchaseOrderType"/>
</schema>';
begin
  dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
/
create table po_tab(
  id number,
  po sys.XMLTYPE
)
XMLTYPE column po
  XMLSCHEMA "http://www.oracle.com/PO.xsd"
  element "PurchaseOrder";
insert into po_tab values(
  1,
  XMLTYPE(
    '<PurchaseOrder xmlns="http://www.oracle.com/PO.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.oracle.com/PO.xsd">
```

```
      <PONum>1001</PONum>
      <Company>Oracle Corp</Company>
    </PurchaseOrder>'
  )
);
```

ODAC supports schema-based XML documents processing. You can open TOraQuery and get TOraXML object with schema-based document the same way as for LOB-based XMLTYPE. You can get value document calling TOraXMLField.AsString or TOraXML.AsString. Extract, Exists, Transform, GetSchema, Validate, and IsSchemaBased functions of TOraXML are available for this XML document type.

```
with OraQuery1 do begin
  RetDoc := TOraXML.Create();
  RetDoc.OCISvcCtx := OraSession1.OCISvcCtx;
  try
    with TOraXMLField(FieldByName('XMLTYPE')).AsXML do begin
      GetSchema(RetDoc, SchemaURL, RootElem);
      Str := RetDoc.AsString;
    end;
  finally
    RetDoc.Free;
  end;
end;
```

See Also

- TOraXML
- TOraXMLField
- TOraLob

Request Support DAC Forum Provide Feedback

4.19.5 VARRAY Data Type

Everything considered in Working with objects is right for Arrays. Some problems appear when you need to use large arrays in dataset. As ODAC creates one field for each array item great number of TField objects are created. As a result the perfomance decreases. So ODAC has the limitation and creates fields for first 1000 items. However you can access all items with TOraArray object. Another way is to set TOraQuery.SparseArray to True and access array items by TArrayField object.

If such types are created

```
CREATE TYPE TODACArray1 AS VARRAY (5) OF NUMBER;
```

```
CREATE TYPE TODACArray2 AS VARRAY (4) OF CHAR(10);
CREATE TABLE ODAC_Array (
  Code NUMBER,
  Title VARCHAR2(10),
  Arr1 TODACArray1,
  Arr2 TODACArray2,
);
```

To access array items you can call FieldByName method. For example

```
Value := Query.FieldByName('Arr1[0]').AsInteger;
```

If ObjectField property is True this code is correct

```
Value := TArrayField(Query.FieldByName('Arr1')).Fields[0].AsInteger;
```

Using TOraDataSet.GetArray you can access array items through TOraArray object

```
Value:= Query.GetArray('Arr1').ItemAsInteger[0];
```

You can use VARRAY type for parameters of SQL and PL/SQL statements. You need to assign dtArray to TOraParam.DataType and use TOraParam.AsArray property to access array items.

For example:

```
var
  OraSQL: TOraSQL;
. . .
OraSQL.SQL.Text := 'INSERT INTO ODAC_Array (Code, Arr1, Arr2)' +
                   'VALUES (:Code, :Arr1, :Arr2)';
OraSQL.ParamByName('Code').AsInteger := 10;
with OraSQL.ParamByName('Arr1').AsArray do begin
  AllocObject(OraSession.OCISvcCtx, 'TODACArray1');
  ItemAsInteger[0] := 12;
  AttrAsInteger['[1]'] := 10;
  ItemAsInteger[3] := 133;
end;
with OraSQL. ParamByName('Arr2').AsArray do begin
  OCISvcCtx:= OraSession.OCISvcCtx;
  AllocObject('TODACArray2');
  AttrAsString['[2]']:= 'eeee';
  ItemAsString[0]:= 'FFFFF';
end;
OraSQL.Execute;
```

See Also

- TOraArray
- Array demo

Reserved.

4.19.6 DML Array

Using of array binding feature can greatly speed up the execution of the application on insert or update big volume of data. The main advantage is that array binding allows you to execute several INSERT SQL statements with the different parameters at once. Note that you access server only once - that increases speed of update a lot.

The following is a sample of using DML Array.

The following table is used in this sample.

```
CREATE TABLE dept
(
  deptno NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
  dname VARCHAR2(14),
  loc VARCHAR2(13)
);
```

At first, you should open a session:

```
  OraSession.UserName := 'scott';
  OraSession.Password := 'tiger';
  OraSession.Server := 'Ora';
  OraSession.Connect;
```

After that you should specify SQL statement for the execution:

```
  OraSQL.SQL.Text := 'INSERT INTO dept VALUES(:deptno_p, :dname_p, :loc_p)';
```

Colons in the SQL text mean parameters with the values which will be specified later.

Now you should specify parameter type for each parameter from the SQL text and the Length property of the parameters, which should be equal to the number of SQL statement executions.

```
  OraSQL.ParamByName('deptno_p').DataType := ftInteger;
  OraSQL.ParamByName('deptno_p').Length := 4;

  OraSQL.ParamByName('dname_p').DataType := ftString;
  OraSQL.ParamByName('dname_p').Length := 4;

  OraSQL.ParamByName('loc_p').DataType := ftString;
  OraSQL.ParamByName('loc_p').Length := 4;
```

You should call Prepare method before specifying values for the highest efficiency.

```
  OraSQL.Prepare;
```

Each item of the array must correspond to the single execution of the SQL statement.

```
  OraSQL.ParamByName('deptno_p').ItemAsInteger[1] := 10;
  OraSQL.ParamByName('dname_p').ItemAsString[1] := 'ACCOUNTING';
  OraSQL.ParamByName('loc_p').ItemAsString[1] := 'NEW YORK';

  OraSQL.ParamByName('deptno_p').ItemAsInteger[2] := 20;
  OraSQL.ParamByName('dname_p').ItemAsString[2] := 'RESEARCH';
  OraSQL.ParamByName('loc_p').ItemAsString[2] := 'DALLAS';

  OraSQL.ParamByName('deptno_p').ItemAsInteger[3] := 30;
  OraSQL.ParamByName('dname_p').ItemAsString[3] := 'SALES';
  OraSQL.ParamByName('loc_p').ItemAsString[3] := 'CHICAGO';

  OraSQL.ParamByName('deptno_p').ItemAsInteger[4] := 40;
  OraSQL.ParamByName('dname_p').ItemAsString[4] := 'OPERATIONS';
  OraSQL.ParamByName('loc_p').ItemAsString[4] := 'BOSTON';
```

After accomplishing previous steps you should call Execute method that assumes a parameter specifying how many times SQL statement will be executed. Note that the value of this method argument must be equal to the number of parameters value elements. Now you can execute SELECT * FROM Dept with any Oracle tool (you can use dbForge Studio for Oracle for this purpose) and see four new records appended.

```
  OraSQL.Execute(4);
```


Request Support DAC Forum Provide Feedback

4.19.7 Cursors

Since Oracle 7.3 the REF CURSOR type has been available to allow recordsets to be returned from stored procedures and functions.

Oracle 9i introduced the predefined SYS_REFCURSOR type, meaning we no longer have to define our own REF CURSOR types.

Using Ref Cursors

The example below uses a ref cursor to return a subset of records in the EMP table.

The following procedure opens a query using a SYS_REFCURSOR output parameter. Notice the cursor is not closed in the procedure. It is up to the calling code to manage the cursor once it has been opened.

```
CREATE OR REPLACE PROCEDURE sp_get_emp (p_deptno    IN  emp.deptno%TYPE, p_r
BEGIN
  OPEN p_recordset FOR
    SELECT ename,
           empno,
```

```
            deptno
    FROM    emp
    WHERE   deptno = p_deptno
    ORDER BY ename;
END sp_get_emp;
/
```

Using cursors in ODAC

In ODAC work with cursors may be implemented using the following components:

TOraQuery , TOraStoredProc , TOraSQL .

Below is a sample working with cursors in TOraStoredProc :

Delphi

```
program Project1;
{$APPTYPE CONSOLE}
{$R *.res}
uses
  System.SysUtils, Ora;
var
  OraSession: TOraSession;
  OraStoredProc: TOraStoredProc;
begin
  OraSession := TOraSession.Create(nil);
  try
    OraSession.ConnectString := 'scott/tiger@orc';
    OraSession.Connect;
    WriteLn('Использование TOraStoredProc');
    OraStoredProc := TOraStoredProc.Create(nil);
    try
      OraStoredProc.Session := OraSession;
      OraStoredProc.StoredProcName := 'sp_get_emp';
      OraStoredProc.Prepare;
      OraStoredProc.ParamByName('p_deptno').AsInteger := 10;
      OraStoredProc.Execute;
      while not OraStoredProc.Eof do begin
        Writeln(OraStoredProc.FieldByName('ename').AsString);
        OraStoredProc.Next;
      end;
    finally
      OraStoredProc.Free;
    end;
  finally
    OraSession.Free;
    readln;
  end;
end.
```

C++Builder

```
#include <vcl.h>
#pragma hdrstop
```

```
#include <tchar.h>
#include <stdio.h>
#include <Ora.hpp>
#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
 TOraSession *OraSession = new TOraSession(NULL);
 try
 {
  OraSession->ConnectString = "scott/tiger@orcl1120";
  OraSession->Connect();
  TOraStoredProc *OraStoredProc = new TOraStoredProc(NULL);
  try
  {
   OraStoredProc->StoredProcName = "sp_get_emp";
   OraStoredProc->Prepare();
   OraStoredProc->ParamByName("p_deptno")->AsInteger = 10;
   OraStoredProc->Session = OraSession;
   OraStoredProc->Execute();
   while (!OraStoredProc->Eof)
   {
    printf("%s\n", OraStoredProc->FieldByName("ename")->AsString.t_str());
    OraStoredProc->Next();
   }
  }
  __finally
  {
   OraStoredProc->Free();
  }
 }
 __finally
 {
  OraSession->Free();
  system("pause");
 }
 return 0;
}
```

If several output parameters in the procedure are cursors, then TOraStoredProc will work only with the first one as with a DataSet. To retrieve other DataSets, the asCursor method must be used:

```
OraQuery.Cursor := OraSession.ParamByName('Cur2').AsCursor;
OraQuery.Open;
```


Request Support DAC Forum Provide Feedback

4.19.8 PL/SQL Tables

ODAC allows you to use PL/SQL arrays known as PL/SQL Tables as parameters of anonymous PL/SQL blocks or as parameters of stored procedures. As ordinary arrays, PL/

SQL arrays can be used for storing the same data accessible by index.

We will use standard Dept table in our sample. If you don't have this table at your database see SQL script at Demos\InstallDemoObjects.sql folder. Following sample demonstrates how to update several records from Dept table simultaneously using parameter of PL/SQL Table type.

Here is a PL/SQL block used in our sample:

```
DECLARE
  i INTEGER;
BEGIN
  i:= 1;
  FOR rec IN (SELECT DeptNo FROM Scott.Dept
    WHERE RowNum <= 10 ORDER BY DeptNo)
  LOOP
    UPDATE Scott.Dept
      SET DName = :NameArr(i)
      WHERE DeptNo = Rec.DeptNo;
    i:= i + 1;
  END LOOP;
END;
```

There is one parameter in the text of the sample PL/SQL block with NameArr name. It has PL/SQL Table type. This SQL updates DName field of Dept table with the values from NameArr array.

At first, you should open a session:

```
  OraSession.UserName := 'scott';
  OraSession.Password := 'tiger';
  OraSession.Server := 'Ora';
  OraSession.Connect;
```

After that you should specify SQL statement for the execution:

```
  OraSQL.SQL.Add('DECLARE');
  OraSQL.SQL.Add('  i INTEGER;');
  OraSQL.SQL.Add('BEGIN');
  OraSQL.SQL.Add('  i:= 1;');
  OraSQL.SQL.Add('  FOR rec IN (SELECT DeptNo FROM Scott.Dept');
  OraSQL.SQL.Add('    WHERE RowNum <= 10 ORDER BY DeptNo)');
  OraSQL.SQL.Add('  LOOP');
  OraSQL.SQL.Add('    UPDATE Scott.Dept');
  OraSQL.SQL.Add('       SET DName = :NameArr(i)');
  OraSQL.SQL.Add('       WHERE DeptNo = Rec.DeptNo;');
  OraSQL.SQL.Add('    i:= i + 1;');
  OraSQL.SQL.Add('  END LOOP;');
  OraSQL.SQL.Add('END;');
```

The NameArr parameter value should be specified later.

Then you need to specify that the parameter with NameArr name has PL/SQL Table type. To do it, you should set Table property to True and Length property of the parameter to the required value. If Dept table has four records, the size of the array also must be four.

```
  OraSQL.ParamByName('NameArr').Table := True;
  OraSQL.ParamByName('NameArr').DataType := ftString;
  OraSQL.ParamByName('NameArr').Length := 4;
```

After that you need to set values for the array items of NameArr parameter. The amount of array items must be equal to the value of Length property.

```
  OraSQL.ParamByName('NameArr').ItemAsString[1] := 'London';
  OraSQL.ParamByName('NameArr').ItemAsString[2] := 'Berlin';
  OraSQL.ParamByName('NameArr').ItemAsString[3] := 'Geneva';
  OraSQL.ParamByName('NameArr').ItemAsString[4] := 'Vienna';
```

Now you can execute SQL by calling Execute method of TOraSQL component.

```
  OraSQL.Execute;
```


Request Support DAC Forum Provide Feedback

4.20 Writing Oracle External Procedures with ODAC

External procedure is a procedure stored in a dynamic link library (DLL), or libunit in the case of a Java class method. Different programming languages can be used for external procedures creation - C, C++, Object Pascal, Java. External procedure can be called directly from PL/SQL and SQL. You can use ODAC components for writing external procedures for Oracle database. A small example of external procedure using ODAC components is listed below.

For example, let's create an external procedure that saves LOB to file and stores the file name and the file date in a database. Suppose we have the following table to store file names and dates:

```
CREATE TABLE scott.odac_file_list
(
  id integer PRIMARY KEY,
  file_name VARCHAR2(100),
  file_date TIMESTAMP
)
```

Let's create a DLL ExtProc containing our external procedure add_file.

All external procedures and functions in DLL must be listed in the library exports clause.

Before calling any OCI functions in DLL InitOCI procedure must be called. When OCI is no longer needed FreeOCI procedure must be called.

In declaration of procedure add_file cdecl directive must be used. It is necessary to call OCIExtProcGetEnv function, that returns environment, service context and error handles. A call to OCIExtProcGetEnv function is required to make OCI callbacks to database.

Then we will create a data module with TOraSession and TOraQuery components. TOraSession component can be linked to external procedure service context by assigning service context pointer to OCISvcCtx property of TOraSession. After such assignment we can execute queries through OraSession.

An external procedure must not raise Delphi exceptions. All these exceptions must be processed inside the procedure and procedure can raise PL/SQL exceptions with OCIExtProcRaiseExcpWithMsg OCI function.

The source code of DLL, that contains add_file procedure is the following:

```
library ExtProc;
uses
  SysUtils,
  Classes,
  DB, Ora, OraCall, OraError, OraClasses,
  Data in 'Data.pas' {dmData: TDataModule};
{$R *.res}
procedure add_file(Context: pOCIExtProcContext; Id: pOCINumber;
  FileName: PChar; FileDate: pOCIDateTime; FileText: pOCILobLocator); cdecl;
var
  hEnv: pOCIEnv;
  hSvcCtx: pOCISvcCtx;
  hError: pOCIError;
  dmData: TdmData;
  OraLob: TOraLob;
begin
  try
    // get OCI service context
    Check(OCIExtProcGetEnv(Context, hEnv, hSvcCtx, hError));
    dmData := TdmData.Create(nil);
    try
      // set sevice context handle in OraSession
      dmData.OraSession.OCISvcCtx := hSvcCtx;
      with dmData.OraSQL do begin
        ParamByName('ID').DataType := TFieldType(ftNumber);
        ParamByName('ID').AsNumber.OCINumberPtr := Id;
        ParamByName('FILE_NAME').AsString := FileName;
        ParamByName('FILE_DATE').DataType := ftTimeStamp;
        ParamByName('FILE_DATE').AsTimeStamp.OCIDateTime := FileDate;
        Execute;
        OraLob := TOraLob.Create(hSvcCtx);
        try
```

```
          OraLob.OCILobLocator := FileText;
          OraLob.ReadLob;
          OraLob.SaveToFile(FileName);
        finally
          OraLob.Free;
        end;
      end;
    finally
      dmData.Free;
    end;
  except
    on e: EOraError do
      OCIExtProcRaiseExcpWithMsg(Context, e.ErrorCode, PChar(e.Message), Len
    on e: Exception do
      OCIExtProcRaiseExcpWithMsg(Context, 20000, PChar(e.Message), Length(e.
  end;
end;
exports
  add_file;
begin
  // Load oci.dll and link OCI functions
  InitOCI;
end.
```

To use this external procedure compile the DLL and copy it to Oracle server. The DLL must be copied to ORACLE_HOME\bin (Windows) or ORACLE_HOME/lib (UNIX). See Oracle documentation about making the external procedures agent load external procedure libraries from other paths.

External procedures DLL must be defined with CREATE LIBRARY statement. In our ExternalProc Demo the library is created as follows:

```
CREATE OR REPLACE LIBRARY Scott.ExtProcDemo AS
 'C:\oracle\product\10.2.0\db_1\bin\ExtProc.dll'
```

Note: the path to the DLL passed to CREATE LIBRARY statement is case sensitive.

Then we define the external procedure:

```
CREATE PROCEDURE scott.add_file(
   id NUMBER,
   file_name VARCHAR2,
   file_date TIMESTAMP,
   file_text CLOB
  )
  AS LANGUAGE C
  NAME "add_file"
  LIBRARY scott.ExtProcDemo
  WITH CONTEXT
  PARAMETERS (CONTEXT, id OCINUMBER, file_name STRING, file_date OCIDATETIME
```

The "LANGUAGE C" option shows that it is an external procedure written in the language compatible with the C language call specification. The "NAME" option is the name of the

procedure in the DLL. "WITH CONTEXT" option enables OCI callbacks to the database during an external procedure execution. That means an additional CONTEXT parameter is passed to the procedure. It allows the procedure to use a connection to the database.

Now the add_file procedure can be called from an SQL query.

Note to execute external procedures Oracle Net files listener.ora and tnsnames.ora must be configured for external procedures. See Oracle documentation about the configuration Oracle net for external procedures.

SeeAlso

- ExternalProc demo

Request Support DAC Forum Provide Feedback

4.21 Transparent Application Failover Support

Transparent application failover (TAF) is the ability of applications to automatically reconnect to the database if the connection fails. If the server fails, the connection also fails. The next time the client tries to use the connection to execute a new SQL statement, for example, the operating system displays an error to the client. At this point, the user must log in to the database again. With TAF, however, Oracle automatically obtains a new connection to the database. This allows the user to continue to work using the new connection as if the original connection had never failed. If the client is not involved in a database transaction, then users may not notice the failure of the server. Because this reconnection happens automatically, the client application code may not need changes to use TAF. TAF automatically restores:

- Client-Server Database Connections;
- Users' Database Sessions;
- Executing Commands;
- Open Cursors Used for Fetching;
- Active Transactions;
- Server-Side Program Variables.

Unfortunately, TAF cannot automatically restore some session properties. If the application

issued ALTER SESSION commands, then the application must reissue them after TAF processing is complete

Frequently failure of one instance and failover to another takes time. Because of this delay, you may want to inform users that failover is in progress. Additionally, the session on the initial instance may have received some ALTER SESSION commands. These will not be automatically reissued on the second instance. You may need to reissue these commands on the second instance.

To address such problems, you can use TOraSession.OnFailover event. The event is raised during the session recovery process when connection is lost. When connection failure is detected TOnFailover event is raised first time. Then application keeps raising it until connection is restored or user stops failover process.

Transparent Application Failover Restrictions:

- All PL/SQL package states on the server are lost at failover.
- ALTER SESSION statements are lost.
- If failover occurs when a transaction is in process, then each subsequent call causes an error message until the user issues Rollback call. Then a success message is issued. Be sure to check this informational message to see if you must perform any additional operations.
- Continuing work on failed over cursors may cause an error message.
- If the first command after failover is not a SELECT statement or fetch operation, an error message results.
- Failover only takes effect for Oracle 8.0 or higher.
- At failover time, any queries in progress are reissued and processed again from the beginning. This may result in the next query taking a long time if the original query took a long time.

Preparing and Running the Sample

The tnsnames.ora file should be suitably modified for your database entry so that TAF tries to reconnect when the database connection is lost. The tnsnames.ora file is located in the <Oracle_Home>/network/Admin directory. Your database TNS entry should look like this :

```
<DBFAILOVER.US.ORACLE.COM> =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = <myhostname>)(PORT = <1521>))
    )
    (CONNECT_DATA =
            (SERVICE_NAME = <dbfailover>)
      (FAILOVER_MODE = (TYPE = SELECT) (METHOD = BASIC) (RETRIES=100)
      (DELAY=1))
    )
  )
```

where <Oracle_Home> is the directory where your database or SQL* Plus client is installed. Replace the values for the database parameters highlighted in bold with your database parameters.

Build and run the Query project from the ODAC demos. (Please ensure to perform the following steps). Set TOraSession.Options.UseOCI7 to False. Write TOraSession.OnFailover event as follows.

```
procedure TfmMain.OraSessionFailover(Sender: TObject;
  FailoverState: TFailoverState; FailoverType: TFailoverType;
  var Retry: Boolean);
begin
  case FailoverState of
    fsBegin: begin
      ShowMessage('Failover Begin');
      StatusBar1.Panels[0].Text := 'Trying to reconnect, Please wait...';
    end;
    fsAbort: begin
      ShowMessage('Failover Aborted');
      StatusBar1.Panels[0].Text := 'Failover Aborted';
    end;
    fsEnd:
      ShowMessage('Failover End');
    fsError: begin
      StatusBar1.Panels[0].Text := 'Failover Error. Retrying to connect ' +
        'to database. Please wait... ';
      Retry:=true;
    end;
    fsReauth: begin
      ShowMessage('Failover reauthenticating');
      StatusBar1.Panels[0].Text := 'Failover reauthenticating';
    end;
  else
    ShowMessage('Bad Failover');
    StatusBar1.Panels[0].Text := 'Bad Failover';
  end;
end;
```

When run, the sample shows a form with a blank data grid. The user should click "Open" button to start fetching the Dept table records.

For demonstrating TAF, user should restart the database from SQL* Plus using following command:

- To login as a DBA user type

```
SQL> Connect sys/<your_sys_password>@<Your_TNSName> as sysdba
```

- To shutdown and restart database type

```
SQL> startup force
```

After restarting the database, the user should return to the application and refresh the data by clicking "RefreshRecords". The Failover event is called and the Failover handler method displays the appropriate messages in a message box and in the status bar of application. The query will be executed again against the database using a new connection, data fetched and displayed in the data grid.

See Also

- TOraSession.OnFailover

Request Support DAC Forum Provide Feedback

4.22 DBMonitor

To extend monitoring capabilities of ODAC applications there is an additional tool called DBMonitor. It is provided as an alternative to Borland SQL Monitor which is also supported by ODAC.

DBMonitor is an easy-to-use tool to provide visual monitoring of your database applications.

DBMonitor has the following features:

- multiple client processes tracing;
- SQL event filtering (by sender objects);
- SQL parameter and error tracing.

DBMonitor is intended to hamper an application being monitored as little as possible.

To trace your application with DB Monitor you should follow these steps:

- drop TOraSQLMonitor component onto the form;
- turn moDBMonitor option on;
- set to True the Debug property for components you want to trace;
- start DBMonitor before running your program.

 Request Support DAC Forum Provide Feedback

4.23 Writing GUI Applications with ODAC

Since version 3.80 ODAC GUI part is standalone. This means that to make GUI elements such as SQL cursors, connect form, connect dialog etc. available, you should explicitly include OdacVcl unit in your application. This feature is needed for writing console applications.

Delphi and C++Builder

By default ODAC does not require Forms, Controls and other GUI related units. Only TConnectDialog, TOraErrorHandler and TOraAlerter components require the Forms unit.

 Request Support DAC Forum Provide Feedback

4.24 Compatibility with Previous Versions

We always try to keep ODAC compatible with previous versions, but sometimes we have to change the behaviour of ODAC in order to enhance its functionality, or avoid bugs. This topic describes such changes, and how to revert the old ODAC behaviour. We strongly recommend not to turn on the old behaviour of ODAC. Use options described below only if changes applied to ODAC crashed your existent application.

Values of the options described below should be assigned in the **initialization** section of one of the units in your project.

DBAccess.BaseSQLOldBehavior:

The BaseSQL property is similar to the SQL property, but it does not store changes made by AddWhere, DeleteWhere, and SetOrderBy methods. After assigning an SQL text and

modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in ODAC 5.55.1.26. To restore old behavior, set the BaseSQLOldBehavior variable to True.

DBAccess.SQLGeneratorCompatibility:

If the manually assigned RefreshSQL property contains only "WHERE" clause, ODAC uses the value of the BaseSQL property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions AddWhere, DeleteWhere are not taken into account. This behavior was changed in ODAC 6.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

MemDS.SendDataSetChangeEventAfterOpen:

Starting with ODAC 6.20.0.11, the DataSetChange event is sent after the dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

MemDS.DoNotRaiseExcetionOnUaFail:

Starting with ODAC 6.20.0.12, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

Ora.OraQueryCompatibilityMode:

Before ODAC 6, TOraQuery could be editable only when InsertSQL, UpdateSQL, and DeleteSQL properties are assigned. The ability to generate update SQL statements with TOraQuery automatically was added in ODAC 6.00.0.4. Therefore, after upgrading your ODAC to the sixth version, all TOraQuery components in you project become editable, and can be modified by the end users. To restore the old behavior, set the OraQueryCompatibilityMode variable to True.

4.25 Oracle Package Wizard

Oracle Package Wizard is designed for creating wrapper classes for PL/SQL Packages. It greatly simplifies working with types and stored procedures containing in PL/SQL Packages.

Oracle Package Wizard supports:

- All native Oracle types.
- PL/SQL tables of any simple data type except boolean.
- PL/SQL records, including nested records.

To create a wrapper class, perform the following steps:

1. Run Oracle Package Wizard from the ODAC menu.
2. Assign properties to connect to your Oracle server.
3. Choose the packages you want to be wrapped.

Note that items with unsupported parameter types cannot be selected. They are grayed out.

4. Select code generation options:

Parameter type and method conventions

Use Numbers - when this option is checked, Wizard maps Oracle numbers with the precision larger than 15 to ftNumber. Otherwise, they are mapped to ftFloat.
Use Integers - when this option is enabled, Wizard maps Oracle numbers with the precision less than 10 to ftInteger. Otherwise, they are mapped to ftFloat or ftNumber.
Use TimeStamps - when this option is enabled, Wizard maps Oracle timestamps to ftTimeStamp, ftTimeStampTZ, or ftTimeStampLTZ. Otherwise, timestamps are mapped to ftDateTime.
Use DataSets - when this option is enabled, Wizard uses TOraDataSet parameters to return Orcale cursors. Otherwise TOraCursor parameters are used.
Use Unicode - when this option is enabled, Wizard creates fields of the ftWideString data type. Otherwise, ftString is used.
Use variants as parameters - when this option is enabled, variants are used for all simple parameter types.
Generate overloaded methods - when this option is enabled, overloaded methods are created. Otherwise, overloaded subprograms are mapped to the methods with different

suffixes (1, 2, 3 and so on).

Identifier generation rules

Unchangedcase, CapitalizedCase, lowercase, UPPERCASE - these alternative options define character case in identifier names.
Remove underscores - when this option is enabled, Wizard removes underscores from generated identifiers.
Prefix objects with T - when this option is enabled, generated class names are prefixed with 'T'.
Prefix parameters with A - when this option is enabled, method parameters are prefixed with 'A'.

Target environment

Generate code for all versions of Delphi - when this option is enabled, generated code is compatible with the following Delphi versions: Delphi 6, Delphi 7, Borland Developer Studio 2006, CodeGear Delphi 2007 for Win32. Otherwise, generated code will work surely only in the current version of Delphi.
Generated code for - select Win32, CLR or Both to determine environments that generated code will be compatible with.

5. Define files to be generated:
   - Select the target directory.
   - Specify the unit name.
   - Enable the "Add to project" option if you want to add the generated unit to the current project.
   - Enable the "Generate as components" option if you want to generate components registration code.
   - Choice the Component palette tab name that will be used for generating components registration code.
   - Enable the "Generate resources" option to generate resource files (*.res files for Win32 and *.bmp files for CLR). In case of a CLR code generation, you must specify the Images subdirectory name.

Press the Generate button to generate classes for selected packages.

Request Support DAC Forum Provide Feedback

4.26 64-bit Development with Embarcadero RAD Studio XE2

RAD Studio XE2 Overview

RAD Studio XE2 is the major breakthrough in the line of all Delphi versions of this product. It allows deploying your applications both on Windows and Mac OS platforms. Additionally, it is now possible to create 64-bit Windows applications to fully benefit from the power of new hardware. Moreover, you can create visually spectacular applications with the help of the FireMonkey GPU application platform.

Its main features are the following:

- Windows 64-bit platform support;
- Mac OS support;
- FireMonkey application development platform;
- Live data bindings with visual components;
- VCL styles for Windows applications.

Changes in 64-bit Application Development

64-bit platform support implies several important changes that each developer must keep in mind prior to the development of a new application or the modernization of an old one.

General

RAD Studio XE2 IDE is a 32-bit application. It means that it cannot load 64-bit packages at design-time. So, all design-time packages in RAD Studio XE2 IDE are 32-bit.

Therefore, if you develop your own components, you should remember that for the purpose of developing components with the 64-bit platform support, you have to compile run-time packages both for the 32- and 64-bit platforms, while design-time packages need to be compiled only for the 32-bit platform. This might be a source of difficulties if your package is simultaneously both a run-time and a design-time package, as it is more than likely that this

package won't be compiled for the 64-bit platform. In this case, you will have to separate your package into two packages, one of which will be used as run-time only, and the other as design-time only.

For the same reason, if your design-time packages require that certain DLLs be loaded, you should remember that design-time packages can be only 32-bit and that is why they can load only 32-bit versions of these DLLs, while at run-time 64-bit versions of the DLLs will be loaded. Correspondingly, if there are only 64-bit versions of the DLL on your computer, you won't be able to use all functions at design-time and, vice versa, if you have only 32-bit versions of the DLLs, your application won't be able to work at run-time.

Extended type

For this type in a 64-bit applications compiler generates SSE2 instructions instead of FPU, and that greatly improves performance in applications that use this type a lot (where data accuracy is needed). For this purpose, the size and precision of Extended type is reduced:

TYPE	32-bit	64-bit
Extended	10 bytes	8 bytes

The following two additional types are introduced to ensure compatibility in the process of developing 32- and 64-bit applications:

Extended80 – whose size in 32-bit application is 10 bytes; however, this type provides the same precision as its 8-byte equivalent in 64-bit applications.

Extended80Rec – can be used to perform low-level operations on an extended precision floating-point value. For example, the sign, the exponent, and the mantissa can be changed separately. It enables you to perform memory-related operations with 10-bit floating-point variables, but not extended-precision arithmetic operations.

Pointer and Integers

The major difference between 32- and 64-bit platforms is the volume of the used memory and, correspondingly, the size of the pointer that is used to address large memory volumes.

TYPE	32-bit	64-bit
Pointer	4 bytes	8 bytes

At the same time, the size of the Integer type remains the same for both platforms:

TYPE	32-bit	64-bit
Integer	4 bytes	4 bytes

That is why, the following code will work incorrectly on the 64-bit platform:

```
Ptr := Pointer(Integer(Ptr) + Offset);
```

While this code will correctly on the 64-bit platform and incorrectly on the 32-bit platform:

```
Ptr := Pointer(Int64(Ptr) + Offset);
```

For this purpose, the following platform-dependent integer type is introduced:

TYPE	32-bit	64-bit
NativeInt	4 bytes	8 bytes
NativeUInt	4 bytes	8 bytes

This type helps ensure that pointers work correctly both for the 32- and 64-bit platforms:

```
Ptr := Pointer(NativeInt(Ptr) + Offset);
```

However, you need to be extra-careful when developing applications for several versions of Delphi, in which case you should remember that in the previous versions of Delphi the NativeInt type had different sizes:

TYPE	Delphi Version	Size
NativeInt	D5	N/A
NativeInt	D6	N/A
NativeInt	D7	8 bytes
NativeInt	D2005	8 bytes
NativeInt	D2006	8 bytes
NativeInt	D2007	8 bytes
NativeInt	D2009	4 bytes
NativeInt	D2010	4 bytes
NativeInt	Delphi XE	4 bytes
NativeInt	Delphi XE2	4 or 8 bytes

Out parameters

Some WinAPIs have OUT parameters of the SIZE_T type, which is equivalent to NativeInt in Delphi XE2. The problem is that if you are developing only a 32-bit application, you won't be

able to pass Integer to OUT, while in a 64-bit application, you will not be able to pass Int64; in both cases you will have to pass NativeInt.

For example:

```
procedure MyProc(out Value: NativeInt);
begin
  Value := 12345;
end;
var
    Value1: NativeInt;
{$IFDEF WIN32}
  Value2: Integer;
{$ENDIF}
{$IFDEF WIN64}
  Value2: Int64;
{$ENDIF}
begin
   MyProc(Value1); // will be compiled;
 MyProc(Value2); // will not be compiled !!!
end;
```

Win API

If you pass pointers to SendMessage/PostMessage/TControl.Perform, the wParam and lParam parameters should be type-casted to the WPARAM/LPARAM type and not to Integer/Longint.

Correct:

```
SendMessage(hWnd, WM_SETTEXT, 0, LPARAM(@MyCharArray));
```

Wrong:

```
SendMessage(hWnd, WM_SETTEXT, 0, Integer(@MyCharArray));
```

Replace SetWindowLong/GetWindowLog with SetWindowLongPtr/GetWindowLongPtr for GWLP_HINSTANCE, GWLP_ID, GWLP_USERDATA, GWLP_HWNDPARENT and GWLP_WNDPROC as they return pointers and handles. Pointers that are passed to SetWindowLongPtr should be type-casted to LONG_PTR and not to Integer/Longint.

Correct:

```
SetWindowLongPtr(hWnd, GWLP_WNDPROC, LONG_PTR(@MyWindowProc));
```

Wrong:

```
SetWindowLong(hWnd, GWL_WNDPROC, Longint(@MyWindowProc));
```

Pointers that are assigned to the TMessage.Result field should use a type-cast to LRESULT

instead of Integer/Longint.

Correct:

```
Message.Result := LRESULT(Self);
```

Wrong:

```
Message.Result := Integer(Self);
```

All TWM...-records for the windows message handlers must use the correct Windows types for the fields:

```
Msg: UINT; wParam: WPARAM; lParam: LPARAM; Result: LRESULT)
```

Assembler

In order to make your application (that uses assembly code) work, you will have to make several changes to it:

- rewrite your code that mixes Pascal code and assembly code. Mixing them is not supported in 64-bit applications;
- rewrite assembly code that doesn't consider architecture and processor specifics.

You can use conditional defines to make your application work with different architectures.

You can learn more about Assembly code here: http://docwiki.embarcadero.com/RADStudio/en/Using_Inline_Assembly_Code You can also look at the following article that will help you to make your application support the 64-bit platform: http://docwiki.embarcadero.com/RADStudio/en/Converting_32-bit_Delphi_Applications_to_64-bit_Windows

Exception handling

The biggest difference in exception handling between Delphi 32 and 64-bit is that in Delphi XE2 64-bit you will gain more performance because of different internal exception mechanism. For 32-bit applications, the Delphi compiler (dcc32.exe) generates additional code that is executed any way and that causes performance loss. The 64-bit compiler (dcc64.exe) doesn't generate such code, it generates metadata and stores it in the PDATA section of an executable file instead.

But in Delphi XE2 64-bit it's impossible to have more than 16 levels of nested exceptions. Having more than 16 levels of nested exceptions will cause a Run Time error.

Debugging

Debugging of 64-bit applications in RAD Studio XE2 is remote. It is caused by the same reason: RAD Studio XE2 IDE is a 32 application, but your application is 64-bit. If you are trying to debug your application and you cannot do it, you should check that the **Include remote debug symbols** project option is enabled.

To enable it, perform the following steps:

1. Open Project Options (in the main menu **Project->Options**).
2. In the Target combobox, select **Debug configuration - 64-bit Windows platform**. If there is no such option in the combobox, right click "Target Platforms" in Project Manager and select **Add platform**. After adding the 64-bit Windows platform, the **Debug configuration - 64-bit Windows platform** option will be available in the Target combobox.
3. Select **Linking** in the left part of the Project Options form.
4. enable the **Include remote debug symbols** option.

After that, you can run and debug your 64-bit application.

To enable remote debugging, perform the following steps:

1. Install Platform Assistant Server (PAServer) on a remote computer. You can find PAServer in the %RAD_Studio_XE2_Install_Directory%\PAServer directory. The setup_paserver.exe file is an installation file for Windows, and the setup_paserver.zip file is an istallation file for MacOS.
2. Run the PAServer.exe file on a remote computer and set the password that will be used to connect to this computer.
3. On a local computer with RAD Studio XE2 installed, right-click the target platform that you want to debug in Project Manager and select **Assign Remote Profile**. Click the **Add** button in the displayed window, input your profile name, click the **Next** button, input the name of a remote computer and the password to it (that you assigned when you started PAServer on a remote computer).

After that, you can test the connection by clicking the **Test Connection** button. If your connection failed, check that your firewalls on both remote and local computers do not block your connection, and try to establish a connection once more. If your connection succeeded, click the Next button and then the Finish button. Select your newly created profile and click

OK.

After performing these steps you will be able to debug your application on a remote computer. You application will be executed on a remote computer, but you will be able to debug it on your local computer with RAD Studio XE2.

For more information about working with Platform Assistant Server, please refer to http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Running_the_Platform_Assistant_on_Windows

Request Support DAC Forum Provide Feedback

4.27 Database Specific Aspects of 64-bit Development

Oracle Connectivity Aspects

OCI mode:

Since at design-time Rad Studio XE 2 works only with x32 libraries and if a connection to the server is needed at design-time, you need to install Oracle Client (x32) regardless of the intended platform. (If the x32 client is needed only for development, you can use only Oracle Instant Client). By default, ODAC use DEFAULT of Oracle Client, that is why, if a x64 client is the default client at design-time, you need to specify a x32 client. To prevent conflicts between different versions of Oracle Client on the end-user side, you can leave the Home property empty, in this case, the default client will be used.

DIRECT mode:

Since there is no need to install Oracle Client for the DIRECT mode, the development of applications for the x64 platform does not differ from the development of application for Windows x86.

Request Support DAC Forum Provide Feedback

5 Reference

This page shortly describes units that exist in ODAC.

Units

Unit Name	Description
CRAccess	This unit contains base classes for accessing databases.
CRBatchMove	This unit contains implementation of the TCRBatchMove component.
CREncryption	This unit contains base classes for data encryption.
CRGrid	This unit contains the TCRDBGrid component.
CRVio	This unit contains classes for HTTP connections.
DAAlerter	This unit contains the base class for the TOraAlerter component.
DADump	This unit contains the base class for the TOraDump component.
DALoader	This unit contains the base class for the TOraLoader component.
DAScript	This unit contains the base class for the TOraScript component.
DASQLMonitor	This unit contains the base class for the TOraSQLMonitor component.
DBAccess	This unit contains base classes for most of the components.
MemData	This unit contains classes for storing data in memory.
MemDS	This unit contains implementation of the TMemDataSet class.

OdacVcl	This unit contains the visual constituent of ODAC.
Ora	This unit contains main components of ODAC.
OraAlerter	This unit contains implementation of the TOraAlerter component.
OraAQ	This unit contains ODAC components for working with Oracle Advanced Queueing.
OraCall	Defines Oracle Call Interface routines.
OraClasses	OraClasses unit defines following data type constants: dtRowId dtCursor dtOraBlob dtOraClob dtBFILE dtCFILE dtLabel dtFixedChar dtUndefined dtTimeStamp dtTimeStampTZ dtTimeStampLTZ dtIntervalYM dtIntervalDS // obsolete dtBLOBLocator = dtOraBlob dtCLOBLocator = dtOraClob
OraConnectionPool	This unit contains the TOraConnectionPoolManager class for managing connection pool.
OraDataTypeMap	Description is not available at the moment.
OraErrHand	This unit contains the TOraErrorHandler component.
OraError	This unit contains the EOraError exception class.
OraLoader	This unit contains implementation of the TOraLoader component.
OraNet	This unit implements the Direct Mode in ODAC.
OraObjects	This unit contains classes for Oracle OBJECT, ARRAY, TABLE and XMLTYPE data

	types.
OraPackage	This unit contains implementation of the TOraPackage component.
OraProvider	This unit contains implementation of the TOraProvider component.
OraScript	This unit contains implementation of the TOraScript component.
OraServices	Description is not available at the moment.
OraSmart	This unit contains the TSmartQuery and TOraTable components.
OraSQLMonitor	This unit contains implementation of the TOraSQLMonitor component.
OraTransaction	This unit contains implementation of the TOraTransaction component.
VirtualDataSet	This unit contains implementation of the TVirtualDataSet component.
VirtualTable	This unit contains implementation of the TVirtualTable component.

Request Support DAC Forum Provide Feedback

5.1 CRAccess

This unit contains base classes for accessing databases.

Classes

Name	Description
TCRCursor	A base class for classes that work with database cursors.

Types

Name	Description
TBeforeFetchProc	This type is used for the TCustomDADataSet.BeforeFetch event.

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.
TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
TCursorState	Used to set cursor state

 Request Support DAC Forum Provide Feedback

5.1.1 Classes

Classes in the **CRAccess** unit.

Classes

Name	Description
TCRCursor	A base class for classes that work with database cursors.

 Request Support DAC Forum Provide Feedback

5.1.1.1 TCRCursor Class

A base class for classes that work with database cursors.

For a list of all members of this type, see TCRCursor members.

Unit

CRAccess

Syntax

```
TCRCursor = class(TSharedObject);
```

Remarks

TCRCursor is a base class for classes that work with database cursors.

Inheritance Hierarchy

TSharedObject

TCRCursor

5.1.1.1.1 Members

TCRCursor class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

5.1.2 Types

Types in the **CRAccess** unit.

Types

Name	Description
TBeforeFetchProc	This type is used for the TCustomDADataSet.BeforeFetch event.

 Request Support DAC Forum Provide Feedback

5.1.2.1 TBeforeFetchProc Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit

CRAccess

Syntax

```
TBeforeFetchProc = procedure (var Cancel: boolean) of object;
```

Parameters

Cancel

True, if the current fetch operation should be aborted.

 Request Support DAC Forum Provide Feedback

5.1.3 Enumerations

Enumerations in the **CRAccess** unit.

Enumerations

Name	Description
TCRIsolationLevel	Specifies how to handle transactions containing database modifications.

TCRTransactionAction	Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
TCursorState	Used to set cursor state

 Request Support DAC Forum Provide Feedback

5.1.3.1 TCRIsolationLevel Enumeration

Specifies how to handle transactions containing database modifications.

Unit

CRAccess

Syntax

```
TCRIsolationLevel = (ilReadCommitted, ilReadUnCommitted,
ilRepeatableRead, ilIsolated, ilSnapshot, ilCustom);
```

Values

Value	Meaning
ilCustom	The parameters of the transaction are set manually in the Params property.
ilIsolated	The most restricted level of transaction isolation. Database server isolates data involved in current transaction by putting additional processing on range locks. Used to put aside all undesired effects observed in the concurrent accesses to the same set of data, but may lead to a greater latency at times of a congested database environment.
ilReadCommitted	Sets isolation level at which transaction cannot see changes made by outside transactions until they are committed. Only dirty reads (changes made by uncommitted transactions) are eliminated by this state of the isolation level. The default value.
ilReadUnCommitted	The most unrestricted level of the transaction isolation. All types of data access interferences are possible. Mainly used for browsing database and to receive instant data with prospective changes.

ilRepeatableRead	Prevents concurrent transactions from modifying data in the current uncommitted transaction. This level eliminates dirty reads as well as nonrepeatable reads (repeatable reads of the same data in one transaction before and after outside transactions may have started and committed).
ilSnapshot	Uses row versioning. Provides transaction-level read consistency. A data snapshot is taken when the snapshot transaction starts, and remains consistent for the duration of a transaction.

Request Support DAC Forum Provide Feedback

5.1.3.2 TCRTransactionAction Enumeration

Specifies the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Unit

CRAccess

Syntax

```
TCRTransactionAction = (taCommit, taRollback);
```

Values

Value	Meaning
taCommit	Transaction is committed.
taRollback	Transaction is rolled back.

Request Support DAC Forum Provide Feedback

5.1.3.3 TCursorState Enumeration

Used to set cursor state

Unit

CRAccess

Syntax

```
TCursorState = (csInactive, csOpen, csParsed, csPrepared, csBound,
csExecuteFetchAll, csExecuting, csExecuted, csFetching,
csFetchingAll, csFetched);
```

Values

Value	Meaning
csBound	Parameters bound
csExecuted	Statement successfully executed
csExecuteFetchAll	Set before FetchAll
csExecuting	Statement is set before executing
csFetched	Fetch finished or canceled
csFetching	Set on first
csFetchingAll	Set on the FetchAll start
csInactive	Default state
csOpen	statement open
csParsed	Statement parsed
csPrepared	Statement prepared

5.2 CRBatchMove

This unit contains implementation of the TCRBatchMove component.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

Types

Name	Description
TCRBatchMoveProgressEvent	This type is used for the TCRBatchMove.OnBatchMoveProgress event.

Enumerations

Name	Description
TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

 Request Support DAC Forum Provide Feedback

5.2.1 Classes

Classes in the **CRBatchMove** unit.

Classes

Name	Description
TCRBatchMove	Transfers records between datasets.

 Request Support DAC Forum Provide Feedback

5.2.1.1 TCRBatchMove Class

Transfers records between datasets.

For a list of all members of this type, see TCRBatchMove members.

Unit

CRBatchMove

Syntax

```
TCRBatchMove = class(TComponent);
```

Remarks

The TCRBatchMove component transfers records between datasets. Use it to copy dataset records to another dataset or to delete datasets records that match records in another dataset. The TCRBatchMove.Mode property determines the desired operation type, the TCRBatchMove.Source and TCRBatchMove.Destination properties indicate corresponding datasets.

Note: A TCRBatchMove component is added to the Data Access page of the component palette, not to the ODAC page.

Request Support DAC Forum Provide Feedback

5.2.1.1.1 Members

TCRBatchMove class overview.

Properties

Name	Description
AbortOnKeyViol	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
AbortOnProblem	Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.
ChangedCount	Used to get the number of records changed in the destination dataset.
CommitCount	Used to set the number of records to be batch moved before commit occurs.
Destination	Used to specify the destination dataset for the batch operation.

FieldMappingMode	Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.
KeyViolCount	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
Mappings	Used to set field matching between source and destination datasets for the batch operation.
Mode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
MovedCount	Used to get the number of records that were read from the source dataset during the batch operation.
ProblemCount	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.
RecordCount	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
Source	Used to specify the source dataset for the batch operation.

Methods

Name	Description
Execute	Performs the batch operation.

Events

Name	Description
OnBatchMoveProgress	Occurs when providing feedback to the user about the batch operation in progress is needed.

Request Support DAC Forum Provide Feedback

5.2.1.1.2 Properties

Properties of the **TCRBatchMove** class.

For a complete list of the **TCRBatchMove** class members, see the TCRBatchMove Members topic.

Public

Name	Description
ChangedCount	Used to get the number of records changed in the destination dataset.
KeyViolCount	Used to get the number of records that could not be moved to or from the destination dataset because of integrity or key violations.
MovedCount	Used to get the number of records that were read from the source dataset during the batch operation.
ProblemCount	Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Published

Name	Description

AbortOnKeyViol	Used to specify whether the batch operation should be terminated immediately after key or integrity violation.
AbortOnProblem	Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.
CommitCount	Used to set the number of records to be batch moved before commit occurs.
Destination	Used to specify the destination dataset for the batch operation.
FieldMappingMode	Used to specify the way fields of destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.
Mappings	Used to set field matching between source and destination datasets for the batch operation.
Mode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
RecordCount	Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.
Source	Used to specify the source dataset for the batch operation.

See Also

- TCRBatchMove Class
- TCRBatchMove Class Members

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.1 AbortOnKeyViol Property

Used to specify whether the batch operation should be terminated immediately after key or integrity violation.

Class

TCRBatchMove

Syntax

```
property AbortOnKeyViol: boolean default True;
```

Remarks

Use the AbortOnKeyViol property to specify whether the batch operation is terminated immediately after key or integrity violation.

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.2 AbortOnProblem Property

Used to specify whether the batch operation should be terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

Class

TCRBatchMove

Syntax

```
property AbortOnProblem: boolean default True;
```

Remarks

Use the AbortOnProblem property to specify whether the batch operation is terminated immediately when it is necessary to truncate data to make it fit the specified Destination.

 Request Support DAC Forum Provide Feedback

Reserved.

5.2.1.1.2.3 ChangedCount Property

Used to get the number of records changed in the destination dataset.

Class

TCRBatchMove

Syntax

```
property ChangedCount: Integer;
```

Remarks

Use the ChangedCount property to get the number of records changed in the destination dataset. It shows the number of records that were updated in the bmUpdate or bmAppendUpdate mode or were deleted in the bmDelete mode.

Request Support DAC Forum Provide Feedback

5.2.1.1.2.4 CommitCount Property

Used to set the number of records to be batch moved before commit occurs.

Class

TCRBatchMove

Syntax

```
property CommitCount: integer default 0;
```

Remarks

Use the CommitCount property to set the number of records to be batch moved before the commit occurs. If it is set to 0, the operation will be chunked to the number of records to fit 32 Kb.

Request Support DAC Forum Provide Feedback

5.2.1.1.2.5 Destination Property

Used to specify the destination dataset for the batch operation.

Class

TCRBatchMove

Syntax

```
property Destination: TDataSet;
```

Remarks

Specifies the destination dataset for the batch operation.

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.6 FieldMappingMode Property

Used to specify the way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

Class

TCRBatchMove

Syntax

```
property FieldMappingMode: TCRFieldMappingMode default
mmFieldIndex;
```

Remarks

Specifies in what way fields of destination and source datasets will be mapped to each other if the Mappings list is empty.

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.7 KeyViolCount Property

Used to get the number of records that could not be moved to or from the destination dataset

because of integrity or key violations.

Class

TCRBatchMove

Syntax

```
property KeyViolCount: Integer;
```

Remarks

Use the KeyViolCount property to get the number of records that could not be replaced, added, deleted from the destination dataset because of integrity or key violations.

If AbortOnKeyViol is True, then KeyViolCount will never exceed one, because the operation aborts when the integrity or key violation occurs.

See Also

- AbortOnKeyViol

5.2.1.1.2.8 Mappings Property

Used to set field matching between source and destination datasets for the batch operation.

Class

TCRBatchMove

Syntax

```
property Mappings: TStrings;
```

Remarks

Use the Mappings property to set field matching between the source and destination datasets for the batch operation. By default fields matching is based on their position in the datasets. To map the column ColName in the source dataset to the column with the same name in the destination dataset, use:

ColName

Example

To map a column named SourceColName in the source dataset to the column named DestColName in the destination dataset, use:

```
DestColName=SourceColName
```


Request Support DAC Forum Provide Feedback

5.2.1.1.2.9 Mode Property

Used to set the type of the batch operation that will be executed after calling the Execute method.

Class

TCRBatchMove

Syntax

```
property Mode: TCRBatchMode default bmAppend;
```

Remarks

Use the Mode property to set the type of the batch operation that will be executed after calling the Execute method.

Request Support DAC Forum Provide Feedback

5.2.1.1.2.10 MovedCount Property

Used to get the number of records that were read from the source dataset during the batch operation.

Class

TCRBatchMove

Syntax

```
property MovedCount: Integer;
```

Remarks

Use the MovedCount property to get the number of records that were read from the source dataset during the batch operation. This number includes records that caused key or integrity violations or were trimmed.

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.11 ProblemCount Property

Used to get the number of records that could not be added to the destination dataset because of the field type mismatch.

Class

TCRBatchMove

Syntax

```
property ProblemCount: Integer;
```

Remarks

Use the ProblemCount property to get the number of records that could not be added to the destination dataset because of the field type mismatch.

If AbortOnProblem is True, then ProblemCount will never exceed one, because the operation aborts when the problem occurs.

See Also

- AbortOnProblem

 Request Support DAC Forum Provide Feedback

5.2.1.1.2.12 RecordCount Property

Used to indicate the maximum number of records in the source dataset that will be applied to the destination dataset.

Class

TCRBatchMove

Syntax

```
property RecordCount: Integer default 0;
```

Remarks

Determines the maximum number of records in the source dataset, that will be applied to the destination dataset. If it is set to 0, all records in the source dataset will be applied to the destination dataset, starting from the first record. If RecordCount is greater than 0, up to the RecordCount records are applied to the destination dataset, starting from the current record in the source dataset. If RecordCount exceeds the number of records left in the source dataset, batch operation terminates after reaching last record.

Request Support DAC Forum Provide Feedback

5.2.1.1.2.13 Source Property

Used to specify the source dataset for the batch operation.

Class

TCRBatchMove

Syntax

```
property Source: TDataSet;
```

Remarks

Specifies the source dataset for the batch operation.

Request Support DAC Forum Provide Feedback

5.2.1.1.3 Methods

Methods of the **TCRBatchMove** class.

For a complete list of the **TCRBatchMove** class members, see the TCRBatchMove Members topic.

Public

Name	Description
Execute	Performs the batch operation.

See Also

- TCRBatchMove Class
- TCRBatchMove Class Members

Request Support DAC Forum Provide Feedback

5.2.1.1.3.1 Execute Method

Performs the batch operation.

Class

TCRBatchMove

Syntax

```
procedure Execute;
```

Remarks

Call the Execute method to perform the batch operation.

Request Support DAC Forum Provide Feedback

5.2.1.1.4 Events

Events of the **TCRBatchMove** class.

For a complete list of the **TCRBatchMove** class members, see the TCRBatchMove Members topic.

Published

Name	Description
OnBatchMoveProgress	Occurs when providing feedback to the user about the batch operation in progress is needed.

See Also

- TCRBatchMove Class
- TCRBatchMove Class Members

 Request Support DAC Forum Provide Feedback

5.2.1.1.4.1 OnBatchMoveProgress Event

Occurs when providing feedback to the user about the batch operation in progress is needed.

Class

TCRBatchMove

Syntax

```
property OnBatchMoveProgress: TCRBatchMoveProgressEvent;
```

Remarks

Write the OnBatchMoveProgress event handler to provide feedback to the user about the batch operation progress.

 Request Support DAC Forum Provide Feedback

5.2.2 Types

Types in the **CRBatchMove** unit.

Types

Name	Description
TCRBatchMoveProgressEvent	This type is used for the TCRBatchMove.OnBatchMoveProgress event.

 Request Support DAC Forum Provide Feedback

5.2.2.1 TCRBatchMoveProgressEvent Procedure Reference

This type is used for the TCRBatchMove.OnBatchMoveProgress event.

Unit

CRBatchMove

Syntax

```
TCRBatchMoveProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

Percentage of the batch operation progress.

 Request Support DAC Forum Provide Feedback

5.2.3 Enumerations

Enumerations in the **CRBatchMove** unit.

Enumerations

Name	Description

TCRBatchMode	Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.
TCRFieldMappingMode	Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

 Request Support DAC Forum Provide Feedback

5.2.3.1 TCRBatchMode Enumeration

Used to set the type of the batch operation that will be executed after calling the TCRBatchMove.Execute method.

Unit

CRBatchMove

Syntax

```
TCRBatchMode = (bmAppend, bmUpdate, bmAppendUpdate, bmDelete);
```

Values

Value	Meaning
bmAppend	Appends the records from the source dataset to the destination dataset. The default mode.
bmAppendUpdate	Replaces records in the destination dataset with the matching records from the source dataset. If there is no matching record in the destination dataset, the record will be appended to it.
bmDelete	Deletes records from the destination dataset if there are matching records in the source dataset.
bmUpdate	Replaces records in the destination dataset with the matching records from the source dataset.

 Request Support DAC Forum Provide Feedback

5.2.3.2 TCRFieldMappingMode Enumeration

Used to specify the way fields of the destination and source datasets will be mapped to each other if the TCRBatchMove.Mappings list is empty.

Unit

CRBatchMove

Syntax

```
TCRFieldMappingMode = (mmFieldIndex, mmFieldName);
```

Values

Value	Meaning
mmFieldIndex	Specifies that the fields of the destination dataset will be mapped to the fields of the source dataset by field index.
mmFieldName	Mapping is performed by field names.

5.3 CREncryption

This unit contains base classes for data encryption.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption in a client application using various encryption algorithms.

Enumerations

Name	Description
TCREncDataHeader	Specifies whether the additional information is stored with the encrypted data.

TCREncryptionAlgorithm	Specifies the algorithm of data encryption.
TCRHashAlgorithm	Specifies the algorithm of generating hash data.
TCRInvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.

Request Support DAC Forum Provide Feedback

5.3.1 Classes

Classes in the **CREncryption** unit.

Classes

Name	Description
TCREncryptor	The class that performs data encryption and decryption in a client application using various encryption algorithms.

Request Support DAC Forum Provide Feedback

5.3.1.1 TCREncryptor Class

The class that performs data encryption and decryption in a client application using various encryption algorithms.

For a list of all members of this type, see TCREncryptor members.

Unit

CREncryption

Syntax

```
TCREncryptor = class(TComponent);
```


Request Support DAC Forum Provide Feedback

Reserved.

5.3.1.1.1 Members

TCREncryptor class overview.

Properties

Name	Description
DataHeader	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm	Specifies the algorithm of data encryption.
HashAlgorithm	Specifies the algorithm of generating hash data.
InvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.
Password	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description
SetKey	Sets a key, using which data is encrypted.

Request Support DAC Forum Provide Feedback

5.3.1.1.2 Properties

Properties of the **TCREncryptor** class.

For a complete list of the **TCREncryptor** class members, see the TCREncryptor Members topic.

Published

Name	Description

DataHeader	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm	Specifies the algorithm of data encryption.
HashAlgorithm	Specifies the algorithm of generating hash data.
InvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.
Password	Used to set a password that is used to generate a key for encryption.

See Also

- TCREncryptor Class
- TCREncryptor Class Members

 Request Support DAC Forum Provide Feedback

5.3.1.1.2.1 DataHeader Property

Specifies whether the additional information is stored with the encrypted data.

Class

TCREncryptor

Syntax

```
property DataHeader: TCREncDataHeader default ehTagAndHash;
```

Remarks

Use DataHeader to specify whether the additional information is stored with the encrypted data. Default value is ehTagAndHash.

 Request Support DAC Forum Provide Feedback

5.3.1.1.2.2 EncryptionAlgorithm Property

Specifies the algorithm of data encryption.

Class

TCREncryptor

Syntax

```
property EncryptionAlgorithm: TCREncryptionAlgorithm default
eaBlowfish;
```

Remarks

Use EncryptionAlgorithm to specify the algorithm of data encryption. Default value is eaBlowfish.

5.3.1.1.2.3 HashAlgorithm Property

Specifies the algorithm of generating hash data.

Class

TCREncryptor

Syntax

```
property HashAlgorithm: TCRHashAlgorithm default haSHA1;
```

Remarks

Use HashAlgorithm to specify the algorithm of generating hash data. This property is used only if hash is stored with the encrypted data (the DataHeader property is set to ehTagAndHash). Default value is haSHA1.

5.3.1.1.2.4 InvalidHashAction Property

Specifies the action to perform on data fetching when hash data is invalid.

Class

TCREncryptor

Syntax

```
property InvalidHashAction: TCRInvalidHashAction default ihFail;
```

Remarks

Use InvalidHashAction to specify the action to perform on data fetching when hash data is invalid. This property is used only if hash is stored with the encrypted data (the DataHeader property is set to ehTagAndHash). Default value is ihFail.

If the DataHeader property is set to ehTagAndHash, then on data fetching from a server the hash check is performed for each record. After data decryption its hash is calculated and compared with the hash stored in the field. If these values don't coincide, it means that the stored data is incorrect, and depending on the value of the InvalidHashAction property one of the following actions is performed:

ihFail - the EInvalidHash exception is raised and further data reading from the server is interrupted.

ihSkipData - the value of the field for this record is set to Null. No exception is raised.

ihIgnoreError - in spite of the fact that the data is not valid, the value is set in the field. No exception is raised.

Request Support DAC Forum Provide Feedback

5.3.1.1.2.5 Password Property

Used to set a password that is used to generate a key for encryption.

Class

TCREncryptor

Syntax

```
property Password: string stored False;
```

Remarks

Use Password to set a password that is used to generate a key for encryption.

Note: Calling of the SetKey method clears the Password property.

Request Support DAC Forum Provide Feedback

5.3.1.1.3 Methods

Methods of the **TCREncryptor** class.

For a complete list of the **TCREncryptor** class members, see the TCREncryptor Members topic.

Public

Name	Description
SetKey	Sets a key, using which data is encrypted.

See Also

- TCREncryptor Class
- TCREncryptor Class Members

Request Support DAC Forum Provide Feedback

5.3.1.1.3.1 SetKey Method

Sets a key, using which data is encrypted.

Class

TCREncryptor

Syntax

```
procedure SetKey(const Key; Count: Integer); overload;procedure
SetKey(const Key: TBytes; Offset: Integer; Count: Integer);
overload;
```

Parameters

Key

Holds bytes that represent a key.

Offset

Offset in bytes to the position, where the key begins.

Count

Number of bytes to use from Key.

Remarks

Use SetKey to set a key, using which data is encrypted.

Note: Calling of the SetKey method clears the Password property.

 Request Support DAC Forum Provide Feedback

5.3.2 Enumerations

Enumerations in the **CREncryption** unit.

Enumerations

Name	Description
TCREncDataHeader	Specifies whether the additional information is stored with the encrypted data.
TCREncryptionAlgorithm	Specifies the algorithm of data encryption.
TCRHashAlgorithm	Specifies the algorithm of generating hash data.
TCRInvalidHashAction	Specifies the action to perform on data fetching when hash data is invalid.

 Request Support DAC Forum Provide Feedback

5.3.2.1 TCREncDataHeader Enumeration

Specifies whether the additional information is stored with the encrypted data.

Unit

CREncryption

Syntax

```
TCREncDataHeader = (ehTagAndHash, ehTag, ehNone);
```

Values

Value	Meaning
ehNone	No additional information is stored.
ehTag	GUID and the random initialization vector are stored with the encrypted data.
ehTagAndHash	Hash, GUID, and the random initialization vector are stored with the encrypted data.

5.3.2.2 TCREncryptionAlgorithm Enumeration

Specifies the algorithm of data encryption.

Unit

CREncryption

Syntax

```
TCREncryptionAlgorithm = (eaTripleDES, eaBlowfish, eaAES128,
eaAES192, eaAES256, eaCast128, eaRC4);
```

Values

Value	Meaning
eaAES128	The AES encryption algorithm with key size of 128 bits is used.
eaAES192	The AES encryption algorithm with key size of 192 bits is used.
eaAES256	The AES encryption algorithm with key size of 256 bits is used.

eaBlowfish	The Blowfish encryption algorithm is used.
eaCast128	The CAST-128 encryption algorithm with key size of 128 bits is used.
eaRC4	The RC4 encryption algorithm is used.
eaTripleDES	The Triple DES encryption algorithm is used.

Request Support DAC Forum Provide Feedback

5.3.2.3 TCRHashAlgorithm Enumeration

Specifies the algorithm of generating hash data.

Unit

CREncryption

Syntax

```
TCRHashAlgorithm = (haSHA1, haMD5);
```

Values

Value	Meaning
haMD5	The MD5 hash algorithm is used.
haSHA1	The SHA-1 hash algorithm is used.

Request Support DAC Forum Provide Feedback

5.3.2.4 TCRInvalidHashAction Enumeration

Specifies the action to perform on data fetching when hash data is invalid.

Unit

CREncryption

Syntax

```
TCRInvalidHashAction = (ihFail, ihSkipData, ihIgnoreError);
```

Values

Value	Meaning
ihFail	The EInvalidHash exception is raised and further data reading from the server is interrupted.
ihIgnoreError	In spite of the fact that the data is not valid, the value is set in the field. No exception is raised.
ihSkipData	The value of the field for this record is set to Null. No exception is raised.

Request Support DAC Forum Provide Feedback

5.4 CRVio

This unit contains classes for HTTP connections.

Classes

Name	Description
THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to establish an HTTP connection through a proxy server.

Enumerations

Name	Description
TIPVersion	Specifies Internet Protocol version.

Request Support DAC Forum Provide Feedback

5.4.1 Classes

Classes in the **CRVio** unit.

Classes

Name	Description

THttpOptions	This class is used to establish an HTTP connection.
TProxyOptions	This class is used to establish an HTTP connection through a proxy server.

 Request Support DAC Forum Provide Feedback

5.4.1.1 THttpOptions Class

This class is used to establish an HTTP connection.

For a list of all members of this type, see THttpOptions members.

Unit

CRVio

Syntax

```
THttpOptions = class(TPersistent);
```

Remarks

The THttpOptions class is used to establish an HTTP connection.

For more information about HTTP tunneling, see Network Tunneling .

See Also

- Network Tunneling

 Request Support DAC Forum Provide Feedback

5.4.1.1.1 Members

THttpOptions class overview.

Properties

Name	Description
Enabled	Enables an HTTP connection.
Password	Holds the password for HTTP authorization.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

Request Support DAC Forum Provide Feedback

5.4.1.1.2 Properties

Properties of the **THttpOptions** class.

For a complete list of the **THttpOptions** class members, see the THttpOptions Members topic.

Public

Name	Description
Enabled	Enables an HTTP connection.
ProxyOptions	Holds a TProxyOptions object that contains settings for a proxy connection.

Published

Name	Description
Password	Holds the password for HTTP authorization.
TrustServerCertificate	Verifies the server certificate during an SSL handshake.
Url	Holds the URL of the PHP

	script for HTTP tunneling.
Username	Holds the username for HTTP authorization.

See Also

- THttpOptions Class
- THttpOptions Class Members

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.1 Enabled Property

Enables an HTTP connection.

Class

THttpOptions

Syntax

```
property Enabled: boolean default False;
```

Remarks

The Enabled property specifies that a connection is established through HTTP.

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.2 Password Property

Holds the password for HTTP authorization.

Class

THttpOptions

Syntax

```
property Password: string;
```

Remarks

The Password property holds the password for the password-protected directory that contains the HTTP tunneling script.

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.3 ProxyOptions Property

Holds a TProxyOptions object that contains settings for a proxy connection.

Class

THttpOptions

Syntax

```
property ProxyOptions: TProxyOptions;
```

Remarks

The ProxyOptions property holds a TProxyOptions object that contains settings for a proxy connection.

If it is necessary to connect to the server that resides in a different network, sometimes the client can only connect to it through a proxy server. In this case, besides the connection string, you have to set up ProxyOptions.

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.4 TrustServerCertificate Property

Verifies the server certificate during an SSL handshake.

Class

THttpOptions

Syntax

```
property TrustServerCertificate: boolean default False;
```

Remarks

The TrustServerCertificate property specifies whether to verify the server certificate during an SSL handshake. When True, the Odac bypasses walking the certificate chain to verify the certificate. The default value is False.

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.5 Url Property

Holds the URL of the PHP script for HTTP tunneling.

Class

THttpOptions

Syntax

```
property Url: string;
```

Remarks

The Url property holds the URL of the PHP script for HTTP tunneling. For example, if the script is located in the server root, the URL can be the following: http://server/tunnel.php.

 Request Support DAC Forum Provide Feedback

5.4.1.1.2.6 Username Property

Holds the username for HTTP authorization.

Class

THttpOptions

Syntax

```
property Username: string;
```

Remarks

The Username property holds the username for the password-protected directory that

contains the HTTP tunneling script.

 Request Support DAC Forum Provide Feedback

5.4.1.2 TProxyOptions Class

This class is used to establish an HTTP connection through a proxy server.

For a list of all members of this type, see TProxyOptions members.

Unit

CRVio

Syntax

```
TProxyOptions = class(TPersistent);
```

Remarks

The TProxyOptions class is used to establish an HTTP connection through a proxy server.

 Request Support DAC Forum Provide Feedback

5.4.1.2.1 Members

TProxyOptions class overview.

Properties

Name	Description
Hostname	Holds the hostname or IP address of the proxy server.
Password	Holds the proxy password.
Port	Holds the port number of the proxy server.
Username	Holds the proxy username.

 Request Support DAC Forum Provide Feedback

5.4.1.2.2 Properties

Properties of the **TProxyOptions** class.

For a complete list of the **TProxyOptions** class members, see the TProxyOptions Members topic.

Published

Name	Description
Hostname	Holds the hostname or IP address of the proxy server.
Password	Holds the proxy password.
Port	Holds the port number of the proxy server.
Username	Holds the proxy username.

See Also

- TProxyOptions Class
- TProxyOptions Class Members

 Request Support DAC Forum Provide Feedback

5.4.1.2.2.1 Hostname Property

Holds the hostname or IP address of the proxy server.

Class

TProxyOptions

Syntax

```
property Hostname: string;
```

Remarks

The Hostname property holds the hostname or IP address of the proxy server.

 Request Support DAC Forum Provide Feedback

Reserved.

5.4.1.2.2.2 Password Property

Holds the proxy password.

Class

TProxyOptions

Syntax

```
property Password: string;
```

Remarks

The Password property holds the proxy password.

 Request Support DAC Forum Provide Feedback

5.4.1.2.2.3 Port Property

Holds the port number of the proxy server.

Class

TProxyOptions

Syntax

```
property Port: integer default 0;
```

Remarks

Use the Port property to specify the port number of the proxy server.

 Request Support DAC Forum Provide Feedback

5.4.1.2.2.4 Username Property

Holds the proxy username.

Class

TProxyOptions

Syntax

```
property Username: string;
```

Remarks

The Username property holds the proxy username.

Request Support DAC Forum Provide Feedback

5.4.2 Enumerations

Enumerations in the **CRVio** unit.

Enumerations

Name	Description
TIPVersion	Specifies Internet Protocol version.

Request Support DAC Forum Provide Feedback

5.4.2.1 TIPVersion Enumeration

Specifies Internet Protocol version.

Unit

CRVio

Syntax

```
TIPVersion = (ivIPv4, ivIPv6, ivIPBoth);
```

Values

Value	Meaning
ivIPBoth	Specifies that either IPv6 or IPv4 Internet Protocol version is used
ivIPv4	Specifies that the IPv4 Internet Protocol version is used

ivIPv6	Specifies that the IPv6 Internet Protocol version is used

Remarks

Note: When the TIPVersion property is set to **ivIPBoth** , a connection attempt is made via IPv6 if it is enabled in the operating system settings. If the connection attempt fails, a new connection attempt is made via IPv4.

See Also

- TOraSessionOptions.IPVersion

5.5 DAAlerter

This unit contains the base class for the TOraAlerter component.

Classes

Name	Description
TDAAlerter	A base class that defines functionality for database event notification.

Types

Name	Description
TAlerterErrorEvent	This type is used for the TDAAlerter.OnError event.

5.5.1 Classes

Classes in the **DAAlerter** unit.

Classes

Name	Description

TDAAlerter	A base class that defines functionality for database event notification.

Request Support DAC Forum Provide Feedback

5.5.1.1 TDAAlerter Class

A base class that defines functionality for database event notification.

For a list of all members of this type, see TDAAlerter members.

Unit

DAAlerter

Syntax

```
TDAAlerter = class(TComponent);
```

Remarks

TDAAlerter is a base class that defines functionality for descendant classes support database event notification. Applications never use TDAAlerter objects directly. Instead they use descendants of TDAAlerter.

The TDAAlerter component allows you to register interest in and handle events posted by a database server. Use TDAAlerter to handle events for responding to actions and database changes made by other applications. To get events, an application must register required events. To do this, set the Events property to the required events and call the Start method. When one of the registered events occurs OnEvent handler is called.

Request Support DAC Forum Provide Feedback

5.5.1.1.1 Members

TDAAlerter class overview.

Properties

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever connection opens.
Connection	Used to specify the connection for TDAAlerter.

Methods

Name	Description
SendEvent	Sends an event with Name and content Message.
Start	Starts waiting process.
Stop	Stops waiting process.

Events

Name	Description
OnError	Occurs if an exception occurs in waiting process

Request Support DAC Forum Provide Feedback

5.5.1.1.2 Properties

Properties of the **TDAAlerter** class.

For a complete list of the **TDAAlerter** class members, see the TDAAlerter Members topic.

Public

Name	Description
Active	Used to determine if TDAAlerter waits for messages.
AutoRegister	Used to automatically register events whenever

	connection opens.
Connection	Used to specify the connection for TDAAlerter.

See Also

- TDAAlerter Class
- TDAAlerter Class Members

 Request Support DAC Forum Provide Feedback

5.5.1.1.2.1 Active Property

Used to determine if TDAAlerter waits for messages.

Class

TDAAlerter

Syntax

```
property Active: boolean default False;
```

Remarks

Check the Active property to know whether TDAlerter waits for messages or not. Set it to True to register events.

See Also

- Start
- Stop

 Request Support DAC Forum Provide Feedback

5.5.1.1.2.2 AutoRegister Property

Used to automatically register events whenever connection opens.

Class

TDAAlerter

Syntax

```
property AutoRegister: boolean default False;
```

Remarks

Set the AutoRegister property to True to automatically register events whenever connection opens.

 Request Support DAC Forum Provide Feedback

5.5.1.1.2.3 Connection Property

Used to specify the connection for TDAAlerter.

Class

TDAAlerter

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection for TDAAlerter.

 Request Support DAC Forum Provide Feedback

5.5.1.1.3 Methods

Methods of the **TDAAlerter** class.

For a complete list of the **TDAAlerter** class members, see the TDAAlerter Members topic.

Public

Name	Description
SendEvent	Sends an event with Name and content Message.

Start	Starts waiting process.
Stop	Stops waiting process.

See Also

- TDAAlerter Class
- TDAAlerter Class Members

 Request Support DAC Forum Provide Feedback

5.5.1.1.3.1 SendEvent Method

Sends an event with Name and content Message.

Class

TDAAlerter

Syntax

```
procedure SendEvent(const EventName: string; const Message: string);
```

Parameters

EventName

Holds the event name.

Message

Holds the content Message of the event.

Remarks

Use SendEvent procedure to send an event with Name and content Message.

 Request Support DAC Forum Provide Feedback

5.5.1.1.3.2 Start Method

Starts waiting process.

Class

TDAAlerter

Syntax

```
procedure Start;
```

Remarks

Call the Start method to run waiting process. After starting TDAAlerter waits for messages with names defined by the Events property.

See Also

- Stop
- Active
- TOraAlerter.OnEvent
- TOraAlerter.OnTimeOut

Request Support DAC Forum Provide Feedback

5.5.1.1.3.3 Stop Method

Stops waiting process.

Class

TDAAlerter

Syntax

```
procedure Stop;
```

Remarks

Call Stop method to end waiting process.

See Also

- Start

Request Support DAC Forum Provide Feedback

Reserved.

5.5.1.1.4 Events

Events of the **TDAAlerter** class.

For a complete list of the **TDAAlerter** class members, see the TDAAlerter Members topic.

Public

Name	Description
OnError	Occurs if an exception occurs in waiting process

See Also

- TDAAlerter Class
- TDAAlerter Class Members

 Request Support DAC Forum Provide Feedback

5.5.1.1.4.1 OnError Event

Occurs if an exception occurs in waiting process

Class

TDAAlerter

Syntax

```
property OnError: TAlerterErrorEvent;
```

Remarks

The OnError event occurs if an exception occurs in waiting process. Alerter stops in this case. The exception can be accessed using the E parameter.

 Request Support DAC Forum Provide Feedback

5.5.2 Types

Types in the **DAAlerter** unit.

Types

Name	Description
TAlerterErrorEvent	This type is used for the TDAAlerter.OnError event.

 Request Support DAC Forum Provide Feedback

5.5.2.1 TAlerterErrorEvent Procedure Reference

This type is used for the TDAAlerter.OnError event.

Unit

DAAlerter

Syntax

```
TAlerterErrorEvent = procedure (Sender: TDAAlerter; E: Exception)
of object;
```

Parameters

Sender

An object that raised the event.

E

Exception object.

 Request Support DAC Forum Provide Feedback

5.6 DADump

This unit contains the base class for the TOraDump component.

Classes

Name	Description

TDADump	A base class that defines functionality for descendant classes that dump database objects to a script.
TDADumpOptions	This class allows setting up the behaviour of the TDADump class.

Types

Name	Description
TDABackupProgressEvent	This type is used for the TDADump.OnBackupProgress event.
TDARestoreProgressEvent	This type is used for the TDADump.OnRestoreProgress event.

 Request Support DAC Forum Provide Feedback

5.6.1 Classes

Classes in the **DADump** unit.

Classes

Name	Description
TDADump	A base class that defines functionality for descendant classes that dump database objects to a script.
TDADumpOptions	This class allows setting up the behaviour of the TDADump class.

 Request Support DAC Forum Provide Feedback

5.6.1.1 TDADump Class

A base class that defines functionality for descendant classes that dump database objects to

a script.

For a list of all members of this type, see TDADump members.

Unit

DADump

Syntax

```
TDADump = class(TComponent);
```

Remarks

TDADump is a base class that defines functionality for descendant classes that dump database objects to a script. Applications never use TDADump objects directly. Instead they use descendants of TDADump.

Use TDADump descedants to dump database objects, such as tables, stored procedures, and functions for backup or for transferring the data to another SQL server. The dump contains SQL statements to create the table or other database objects and/or populate the table.

Request Support DAC Forum Provide Feedback

5.6.1.1.1 Members

TDADump class overview.

Properties

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Debug	Used to display the statement that is being executed and the values and types of its parameters.
Options	Used to specify the behaviour of a TDADump

	component.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

Methods

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.
BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received from the stream.

Events

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when Oracle raises some error on TDADump.Restore.
OnRestoreProgress	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile

	, or TDADump.RestoreFromStream method execution progress.

 Request Support DAC Forum Provide Feedback

5.6.1.1.2 Properties

Properties of the **TDADump** class.

For a complete list of the **TDADump** class members, see the TDADump Members topic.

Public

Name	Description
Connection	Used to specify a connection object that will be used to connect to a data store.
Options	Used to specify the behaviour of a TDADump component.

Published

Name	Description
Debug	Used to display the statement that is being executed and the values and types of its parameters.
SQL	Used to set or get the dump script.
TableNames	Used to set the names of the tables to dump.

See Also

- TDADump Class
- TDADump Class Members

 Request Support DAC Forum Provide Feedback

5.6.1.1.2.1 Connection Property

Used to specify a connection object that will be used to connect to a data store.

Class

TDADump

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

See Also

- TCustomDAConnection

5.6.1.1.2.2 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TDADump

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug

5.6.1.1.2.3 Options Property

Used to specify the behaviour of a TDADump component.

Class

TDADump

Syntax

```
property Options: TDADumpOptions;
```

Remarks

Use the Options property to specify the behaviour of a TDADump component.

Descriptions of all options are in the table below.

Option Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

Request Support DAC Forum Provide Feedback

5.6.1.1.2.4 SQL Property

Used to set or get the dump script.

Class

TDADump

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set the dump script. The SQL property stores script that is executed by the Restore method. This property will store the result of Backup and BackupQuery. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- Restore
- Backup
- BackupQuery

Request Support DAC Forum Provide Feedback

5.6.1.1.2.5 TableNames Property

Used to set the names of the tables to dump.

Class

TDADump

Syntax

```
property TableNames: string;
```

Remarks

Use the TableNames property to set the names of the tables to dump. Table names must be separated with semicolons. If the property is empty, the Backup method will dump all available tables.

See Also

- Backup

5.6.1.1.3 Methods

Methods of the **TDADump** class.

For a complete list of the **TDADump** class members, see the TDADump Members topic.

Public

Name	Description
Backup	Dumps database objects to the TDADump.SQL property.
BackupQuery	Dumps the results of a particular query.
BackupToFile	Dumps database objects to the specified file.
BackupToStream	Dumps database objects to the stream.
Restore	Executes a script contained in the SQL property.
RestoreFromFile	Executes a script from a file.
RestoreFromStream	Executes a script received from the stream.

See Also

- TDADump Class
- TDADump Class Members

 Request Support DAC Forum Provide Feedback

5.6.1.1.3.1 Backup Method

Dumps database objects to the SQL property.

Class

TDADump

Syntax

```
procedure Backup;
```

Remarks

Call the Backup method to dump database objects. The result script will be stored in the SQL property.

See Also

- SQL
- Restore
- BackupToFile
- BackupToStream
- BackupQuery

 Request Support DAC Forum Provide Feedback

5.6.1.1.3.2 BackupQuery Method

Dumps the results of a particular query.

Class

TDADump

Syntax

```
procedure BackupQuery(const Query: string);
```

Parameters

Query

Holds a query used for data selection.

Remarks

Call the BackupQuery method to dump the results of a particular query. Query must be a valid select statement. If this query selects data from several tables, only data of the first table in the from list will be dumped.

See Also

- Restore
- Backup
- BackupToFile
- BackupToStream

Request Support DAC Forum Provide Feedback

5.6.1.1.3.3 BackupToFile Method

Dumps database objects to the specified file.

Class

TDADump

Syntax

```
procedure BackupToFile(const FileName: string; const Query:
string = '');
```

Parameters

FileName

Holds the file name to dump database objects to.

Query
Your query to receive the data for dumping.

Remarks

Call the BackupToFile method to dump database objects to the specified file.

See Also

- RestoreFromStream
- Backup
- BackupToStream

 Request Support DAC Forum Provide Feedback

5.6.1.1.3.4 BackupToStream Method

Dumps database objects to the stream.

Class

TDADump

Syntax

```
procedure BackupToStream(Stream: TStream; const Query: string =
'');
```

Parameters

Stream
Holds the stream to dump database objects to.

Query
Your query to receive the data for dumping.

Remarks

Call the BackupToStream method to dump database objects to the stream.

See Also

- RestoreFromStream
- Backup

- BackupToFile

5.6.1.1.3.5 Restore Method

Executes a script contained in the SQL property.

Class

TDADump

Syntax

```
procedure Restore;
```

Remarks

Call the Restore method to execute a script contained in the SQL property.

See Also

- RestoreFromFile
- RestoreFromStream
- Backup
- SQL

5.6.1.1.3.6 RestoreFromFile Method

Executes a script from a file.

Class

TDADump

Syntax

```
procedure RestoreFromFile(const FileName: string);
overload;procedure RestoreFromFile(const FileName: string;
```

```
Encoding: TEncoding); overload;
```

Parameters

FileName
Holds the file name to execute a script from.

Remarks

Call the RestoreFromFile method to execute a script from the specified file.

See Also

- Restore
- RestoreFromStream
- BackupToFile

 Request Support DAC Forum Provide Feedback

5.6.1.1.3.7 RestoreFromStream Method

Executes a script received from the stream.

Class

TDADump

Syntax

```
procedure RestoreFromStream(Stream: TStream);
```

Parameters

Stream
Holds a stream to receive a script to be executed.

Remarks

Call the RestoreFromStream method to execute a script received from the stream.

See Also

- Restore
- RestoreFromFile

- BackupToStream

 Request Support DAC Forum Provide Feedback

5.6.1.1.4 Events

Events of the **TDADump** class.

For a complete list of the **TDADump** class members, see the TDADump Members topic.

Published

Name	Description
OnBackupProgress	Occurs to indicate the TDADump.Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.
OnError	Occurs when Oracle raises some error on TDADump.Restore.
OnRestoreProgress	Occurs to indicate the TDADump.Restore, TDADump.RestoreFromFile, or TDADump.RestoreFromStream method execution progress.

See Also

- TDADump Class
- TDADump Class Members

 Request Support DAC Forum Provide Feedback

5.6.1.1.4.1 OnBackupProgress Event

Occurs to indicate the Backup, M:Devart.Dac.TDADump.BackupToFile(System.String) or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution progress.

Class

TDADump

Syntax

```
property OnBackupProgress: TDABackupProgressEvent;
```

Remarks

The OnBackupProgress event occurs several times during the dumping process of the Backup, M:Devart.Dac.TDADump.BackupToFile(System.String), or M:Devart.Dac.TDADump.BackupToStream(Borland.Vcl.TStream) method execution and indicates its progress. ObjectName parameter indicates the name of the currently dumping database object. ObjectNum shows the number of the current database object in the backup queue starting from zero. ObjectCount shows the quantity of database objects to dump. Percent parameter shows the current percentage of the current table data dumped, not the current percentage of the entire dump process.

See Also

- Backup
- BackupToFile
- BackupToStream

5.6.1.1.4.2 OnError Event

Occurs when Oracle raises some error on Restore.

Class

TDADump

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

The OnError event occurs when Oracle raises some error on Restore.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaException.

Note: You should add the DAScript module to the 'uses' list to use the OnError event handler.

 Request Support DAC Forum Provide Feedback

5.6.1.1.4.3 OnRestoreProgress Event

Occurs to indicate the Restore, RestoreFromFile, or RestoreFromStream method execution progress.

Class

TDADump

Syntax

```
property OnRestoreProgress: TDARestoreProgressEvent;
```

Remarks

The OnRestoreProgress event occurs several times during the dumping process of the Restore, RestoreFromFile, or RestoreFromStream method execution and indicates its progress. The Percent parameter of the OnRestoreProgress event handler indicates the percentage of the whole restore script execution.

See Also

- Restore
- RestoreFromFile
- RestoreFromStream

 Request Support DAC Forum Provide Feedback

5.6.1.2 TDADumpOptions Class

This class allows setting up the behaviour of the TDADump class.

For a list of all members of this type, see TDADumpOptions members.

Unit

DADump

Syntax

```
TDADumpOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.6.1.2.1 Members

TDADumpOptions class overview.

Properties

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

 Request Support DAC Forum Provide Feedback

5.6.1.2.2 Properties

Properties of the **TDADumpOptions** class.

For a complete list of the **TDADumpOptions** class members, see the TDADumpOptions Members topic.

Published

Name	Description
AddDrop	Used to add drop statements to a script before creating statements.
CompleteInsert	Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.
GenerateHeader	Used to add a comment header to a script.
QuoteNames	Used for TDADump to quote all database object names in generated SQL statements.

See Also

- TDADumpOptions Class
- TDADumpOptions Class Members

5.6.1.2.2.1 AddDrop Property

Used to add drop statements to a script before creating statements.

Class

TDADumpOptions

Syntax

```
property AddDrop: boolean default True;
```

Remarks

Use the AddDrop property to add drop statements to a script before creating statements.

 Request Support DAC Forum Provide Feedback

5.6.1.2.2.2 CompleteInsert Property

Used to explicitly specify the table fields names when generating the INSERT SQL query. The default value is False.

Class

TDADumpOptions

Syntax

```
property CompleteInsert: boolean default False;
```

Remarks

If the CompleteInsert property is set to True, SQL query will include the field names, for example:

```
INSERT INTO dept(deptno, dname, loc) VALUES ('10', 'ACCOUNTING', 'NEW YORK')
```

If False, it won't include the field names, for example:

```
INSERT INTO dept VALUES ('10', 'ACCOUNTING', 'NEW YORK');
```

 Request Support DAC Forum Provide Feedback

5.6.1.2.2.3 GenerateHeader Property

Used to add a comment header to a script.

Class

TDADumpOptions

Syntax

```
property GenerateHeader: boolean default True;
```

Remarks

Use the GenerateHeader property to add a comment header to a script. It contains script generation date, DAC version, and some other information.

 Request Support DAC Forum Provide Feedback

5.6.1.2.2.4 QuoteNames Property

Used for TDADump to quote all database object names in generated SQL statements.

Class

TDADumpOptions

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If the QuoteNames property is True, TDADump quotes all database object names in generated SQL statements.

 Request Support DAC Forum Provide Feedback

5.6.2 Types

Types in the **DADump** unit.

Types

Name	Description
TDABackupProgressEvent	This type is used for the TDADump.OnBackupProgress event.
TDARestoreProgressEvent	This type is used for the TDADump.OnRestoreProgress event.

 Request Support DAC Forum Provide Feedback

Reserved.

5.6.2.1 TDABackupProgressEvent Procedure Reference

This type is used for the TDADump.OnBackupProgress event.

Unit

DADump

Syntax

```
TDABackupProgressEvent = procedure (Sender: TObject; ObjectName:
string; ObjectNum: integer; ObjectCount: integer; Percent:
integer) of object;
```

Parameters

Sender
An object that raised the event.

ObjectName
The name of the currently dumping database object.

ObjectNum
The number of the current database object in the backup queue starting from zero.

ObjectCount
The quantity of database objects to dump.

Percent
The current percentage of the current table data dumped.

Request Support DAC Forum Provide Feedback

5.6.2.2 TDARestoreProgressEvent Procedure Reference

This type is used for the TDADump.OnRestoreProgress event.

Unit

DADump

Syntax

```
TDARestoreProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

The percentage of the whole restore script execution.

 Request Support DAC Forum Provide Feedback

5.7 DALoader

This unit contains the base class for the TOraLoader component.

Classes

Name	Description
TDAColumn	Represents the attributes for column loading.
TDAColumns	Holds a collection of TDAColumn objects.
TDALoader	This class allows loading external data into database.
TDALoaderOptions	Allows loading external data into database.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

 Request Support DAC Forum Provide Feedback

5.7.1 Classes

Classes in the **DALoader** unit.

Classes

Name	Description
TDAColumn	Represents the attributes for column loading.
TDAColumns	Holds a collection of TDAColumn objects.
TDALoader	This class allows loading external data into database.
TDALoaderOptions	Allows loading external data into database.

Request Support DAC Forum Provide Feedback

5.7.1.1 TDAColumn Class

Represents the attributes for column loading.

For a list of all members of this type, see TDAColumn members.

Unit

DALoader

Syntax

```
TDAColumn = class(TCollectionItem);
```

Remarks

Each TDALoader uses TDAColumns to maintain a collection of TDAColumn objects. TDAColumn object represents the attributes for column loading. Every TDAColumn object corresponds to one of the table fields with the same name as its TDAColumn.Name property.

To create columns at design-time use the column editor of the TDALoader component.

See Also

- TDALoader

- TDAColumns

 Request Support DAC Forum Provide Feedback

5.7.1.1.1 Members

TDAColumn class overview.

Properties

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

 Request Support DAC Forum Provide Feedback

5.7.1.1.2 Properties

Properties of the **TDAColumn** class.

For a complete list of the **TDAColumn** class members, see the TDAColumn Members topic.

Published

Name	Description
FieldType	Used to specify the types of values that will be loaded.
Name	Used to specify the field name of loading table.

See Also

- TDAColumn Class
- TDAColumn Class Members

 Request Support DAC Forum Provide Feedback

5.7.1.1.2.1 FieldType Property

Used to specify the types of values that will be loaded.

Class

TDAColumn

Syntax

```
property FieldType: TFieldType default ftString;
```

Remarks

Use the FieldType property to specify the types of values that will be loaded. Field types for columns may not match data types for the corresponding fields in the database table. TDALoader will cast data values to the types of their fields.

Request Support DAC Forum Provide Feedback

5.7.1.1.2.2 Name Property

Used to specify the field name of loading table.

Class

TDAColumn

Syntax

```
property Name: string;
```

Remarks

Each TDAColumn corresponds to one field of the loading table. Use the Name property to specify the name of this field.

See Also

- FieldType

Request Support DAC Forum Provide Feedback

5.7.1.2 TDAColumns Class

Holds a collection of TDAColumn objects.

For a list of all members of this type, see TDAColumns members.

Unit

DALoader

Syntax

```
TDAColumns = class(TOwnedCollection);
```

Remarks

Each TDAColumns holds a collection of TDAColumn objects. TDAColumns maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection. At design-time, use the Columns editor to add, remove, or modify columns.

See Also

- TDALoader
- TDAColumn

 Request Support DAC Forum Provide Feedback

5.7.1.2.1 Members

TDAColumns class overview.

Properties

Name	Description
Items	Used to access individual columns.

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.7.1.2.2 Properties

Properties of the **TDAColumns** class.

For a complete list of the **TDAColumns** class members, see the TDAColumns Members topic.

Public

Name	Description
Items	Used to access individual columns.

See Also

- TDAColumns Class
- TDAColumns Class Members

Request Support DAC Forum Provide Feedback

5.7.1.2.2.1 Items Property(Indexer)

Used to access individual columns.

Class

TDAColumns

Syntax

```
property Items[Index: integer]: TDAColumn; default;
```

Parameters

Index

Holds the Index of TDAColumn to refer to.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of TDAColumn.

See Also

- TDAColumn

 Request Support DAC Forum Provide Feedback

5.7.1.3 TDALoader Class

This class allows loading external data into database.

For a list of all members of this type, see TDALoader members.

Unit

DALoader

Syntax

```
TDALoader = class(TComponent);
```

Remarks

TDALoader allows loading external data into database. To specify the name of loading table set the TDALoader.TableName property. Use the TDALoader.Columns property to access individual columns. Write the TDALoader.OnGetColumnData or TDALoader.OnPutData event handlers to read external data and pass it to the database. Call the TDALoader.Load method to start loading data.

See Also

- TOraLoader Component
- TOraLoader

 Request Support DAC Forum Provide Feedback

5.7.1.3.1 Members

TDALoader class overview.

Properties

Name	Description

Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	See the TOraLoader.Session property.
TableName	Used to specify the name of the table to which data will be loaded.

Methods

Name	Description
CreateColumns	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.
Load	Starts loading data.
LoadFromDataSet	Loads data from the specified dataset.
PutColumnData	Overloaded. Puts the value of individual columns.

Events

Name	Description
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

5.7.1.3.2 Properties

Properties of the **TDALoader** class.

For a complete list of the **TDALoader** class members, see the TDALoader Members topic.

Public

Name	Description
Columns	Used to add a TDAColumn object for each field that will be loaded.
Connection	See the TOraLoader.Session property.
TableName	Used to specify the name of the table to which data will be loaded.

See Also

- TDALoader Class
- TDALoader Class Members

 Request Support DAC Forum Provide Feedback

5.7.1.3.2.1 Columns Property

Used to add a TDAColumn object for each field that will be loaded.

Class

TDALoader

Syntax

```
property Columns: TDAColumns stored IsColumnsStored;
```

Remarks

Use the Columns property to add a TDAColumn object for each field that will be loaded.

See Also

- TDAColumns

 Request Support DAC Forum Provide Feedback

5.7.1.3.2.2 Connection Property

See the TOraLoader.Session property.

Class

TDALoader

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify TCustomDAConnection in which TDALoader will be executed. If Connection is not connected, the Load method calls TCustomDAConnection.Connect.

See Also

- TCustomDAConnection

 Request Support DAC Forum Provide Feedback

5.7.1.3.2.3 TableName Property

Used to specify the name of the table to which data will be loaded.

Class

TDALoader

Syntax

```
property TableName: string;
```

Remarks

Set the TableName property to specify the name of the table to which data will be loaded. Add TDAColumn objects to Columns for the fields that are needed to be loaded.

See Also

- TDAColumn
- TCustomDAConnection.GetTableNames

 Request Support DAC Forum Provide Feedback

5.7.1.3.3 Methods

Methods of the **TDALoader** class.

For a complete list of the **TDALoader** class members, see the TDALoader Members topic.

Public

Name	Description
CreateColumns	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.
Load	Starts loading data.
LoadFromDataSet	Loads data from the specified dataset.
PutColumnData	Overloaded. Puts the value of individual columns.

See Also

- TDALoader Class
- TDALoader Class Members

 Request Support DAC Forum Provide Feedback

5.7.1.3.3.1 CreateColumns Method

Creates TDAColumn objects for all fields of the table with the same name as TableName.

Class

TDALoader

Syntax

```
procedure CreateColumns;
```

Remarks

Call the CreateColumns method to create TDAColumn objects for all fields of the table with the same name as TableName. If columns were created before, they will be recreated. You can call CreateColumns from the component popup menu at design-time. After you can customize column loading by setting properties of TDAColumn objects.

See Also

- TDAColumn
- TableName

Request Support DAC Forum Provide Feedback

5.7.1.3.3.2 Load Method

Starts loading data.

Class

TDALoader

Syntax

```
procedure Load; virtual;
```

Remarks

Call the Load method to start loading data. At first it is necessary to create columns and write one of the OnPutData or OnGetColumnData event handlers.

See Also

- OnGetColumnData
- OnPutData

Request Support DAC Forum Provide Feedback

5.7.1.3.3.3 LoadFromDataSet Method

Loads data from the specified dataset.

Class

TDALoader

Syntax

```
procedure LoadFromDataSet(DataSet: TDataSet);
```

Parameters

DataSet

Holds the dataset to load data from.

Remarks

Call the LoadFromDataSet method to load data from the specified dataset. There is no need to create columns and write event handlers for OnPutData and OnGetColumnData before calling this method.

Request Support DAC Forum Provide Feedback

5.7.1.3.3.4 PutColumnData Method

Puts the value of individual columns.

Class

TDALoader

Overload List

Name	Description
PutColumnData(Col: integer; Row: integer; **const** Value: variant)	Puts the value of individual columns by the column index.
PutColumnData(**const** ColName: **string**; Row: integer; **const** Value: variant)	Puts the value of individual columns by the column name.

Request Support DAC Forum Provide Feedback

Puts the value of individual columns by the column index.

Class

TDALoader

Syntax

```
procedure PutColumnData(Col: integer; Row: integer; const Value: variant); overload; virtual;
```

Parameters

Col

Holds the index of a loading column. The first column has index 0.

Row

Holds the number of loading row. Row starts from 1.

Value

Holds the column value.

Remarks

Call the PutColumnData method to put the value of individual columns. PutColumnData can be only called from the OnPutData event handler. The Col parameter indicates the index of loading column. The first column has index 0. The Row parameter indicates the number of the loading row. Row starts from 1.

This overloaded method works faster because it searches the right index by its index, not by the index name.

The value of a column should be assigned to the Value parameter.

See Also

- TDALoader.OnPutData

 Request Support DAC Forum Provide Feedback

Puts the value of individual columns by the column name.

Class

TDALoader

Syntax

```
procedure PutColumnData(const ColName: string; Row: integer;
const Value: variant); overload;
```

Parameters

ColName
Hods the name of a loading column.

Row
Holds the number of loading row. Row starts from 1.

Value
Holds the column value.

 Request Support DAC Forum Provide Feedback

5.7.1.3.4 Events

Events of the **TDALoader** class.

For a complete list of the **TDALoader** class members, see the TDALoader Members topic.

Public

Name	Description
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.

OnPutData	Occurs when putting loading data by rows is needed.

See Also

- TDALoader Class
- TDALoader Class Members

Request Support DAC Forum Provide Feedback

5.7.1.3.4.1 OnGetColumnData Event

Occurs when it is needed to put column values.

Class

TDALoader

Syntax

```
property OnGetColumnData: TGetColumnDataEvent;
```

Remarks

Write the OnGetColumnData event handler to put column values. TDALoader calls the OnGetColumnData event handler for each column in the loop. Column points to a TDAColumn object that corresponds to the current loading column. Use its Name or Index property to identify what column is loading. The Row parameter indicates the current loading record. TDALoader increments the Row parameter when all the columns of the current record are loaded. The first row is 1. Set EOF to True to stop data loading. Fill the Value parameter by column values. To start loading call the Load method.

Another way to load data is using the OnPutData event.

Example

This handler loads 1000 rows.

```
procedure TfmMain.GetColumnData(Sender: TObject;
   Column: TDAColumn; Row: Integer; var Value: Variant;
   var EOF: Boolean);
begin
```

```
    if Row <= 1000 then begin
      case Column.Index of
        0: Value := Row;
        1: Value := Random(100);
        2: Value := Random*100;
        3: Value := 'abc01234567890123456789';
        4: Value := Date;
      else
        Value := Null;
      end;
    end
    else
      EOF := True;
end;
```

See Also

- OnPutData
- Load

 Request Support DAC Forum Provide Feedback

5.7.1.3.4.2 OnProgress Event

Occurs if handling data loading progress of the LoadFromDataSet method is needed.

Class

TDALoader

Syntax

```
property OnProgress: TLoaderProgressEvent;
```

Remarks

Add a handler to this event if you want to handle data loading progress of the LoadFromDataSet method.

See Also

- LoadFromDataSet

 Request Support DAC Forum Provide Feedback

5.7.1.3.4.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

TDALoader

Syntax

```
property OnPutData: TDAPutDataEvent;
```

Remarks

Write the OnPutData event handler to put loading data by rows.

Note that rows should be loaded from the first in the ascending order.

To start loading, call the Load method. It is more effective way to load data in comparison with using OnGetColumnData. The OnPutData event handler must send column data by the TDALoader.PutColumnData method. TDALoader will flush data to Oracle when it is needed.

 Request Support DAC Forum Provide Feedback

5.7.1.4 TDALoaderOptions Class

Allows loading external data into database.

For a list of all members of this type, see TDALoaderOptions members.

Unit

DALoader

Syntax

```
TDALoaderOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.7.1.4.1 Members

TDALoaderOptions class overview.

Properties

Name	Description
UseBlankValues	Forces ODAC to fill the buffer with null values after loading a row to the database.

 Request Support DAC Forum Provide Feedback

5.7.1.4.2 Properties

Properties of the **TDALoaderOptions** class.

For a complete list of the **TDALoaderOptions** class members, see the TDALoaderOptions Members topic.

Public

Name	Description
UseBlankValues	Forces ODAC to fill the buffer with null values after loading a row to the database.

See Also

- TDALoaderOptions Class
- TDALoaderOptions Class Members

 Request Support DAC Forum Provide Feedback

5.7.1.4.2.1 UseBlankValues Property

Forces ODAC to fill the buffer with null values after loading a row to the database.

Class

TDALoaderOptions

Syntax

```
property UseBlankValues: boolean default True;
```

Remarks

Used to force ODAC to fill the buffer with null values after loading a row to the database.

Request Support DAC Forum Provide Feedback

5.7.2 Types

Types in the **DALoader** unit.

Types

Name	Description
TDAPutDataEvent	This type is used for the TDALoader.OnPutData event.
TGetColumnDataEvent	This type is used for the TDALoader.OnGetColumnData event.
TLoaderProgressEvent	This type is used for the TDALoader.OnProgress event.

Request Support DAC Forum Provide Feedback

5.7.2.1 TDAPutDataEvent Procedure Reference

This type is used for the TDALoader.OnPutData event.

Unit

DALoader

Syntax

```
TDAPutDataEvent = procedure (Sender: TDALoader) of object;
```

Parameters

Sender
An object that raised the event.

 Request Support DAC Forum Provide Feedback

5.7.2.2 TGetColumnDataEvent Procedure Reference

This type is used for the TDALoader.OnGetColumnData event.

Unit

DALoader

Syntax

```
TGetColumnDataEvent = procedure (Sender: TObject; Column:
TDAColumn; Row: integer; var Value: variant; var IsEOF: boolean)
of object;
```

Parameters

Sender
An object that raised the event.

Column
Points to TDAColumn object that corresponds to the current loading column.

Row
Indicates the current loading record.

Value
Holds column values.

IsEOF
True, if data loading needs to be stopped.

 Request Support DAC Forum Provide Feedback

5.7.2.3 TLoaderProgressEvent Procedure Reference

This type is used for the TDALoader.OnProgress event.

Unit

DALoader

Syntax

```
TLoaderProgressEvent = procedure (Sender: TObject; Percent:
integer) of object;
```

Parameters

Sender

An object that raised the event.

Percent

Percentage of the load operation progress.

Request Support DAC Forum Provide Feedback

5.8 DAScript

This unit contains the base class for the TOraScript component.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.
TDAStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDAStatements	Holds a collection of TDAStatement objects.

Types

Name	Description
TAfterStatementExecuteEvent	This type is used for the TDAScript.AfterExecute event.
TBeforeStatementExecuteEvent	This type is used for the TDAScript.BeforeExecute event.
TOnErrorEvent	This type is used for the TDAScript.OnError event.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

 Request Support DAC Forum Provide Feedback

5.8.1 Classes

Classes in the **DAScript** unit.

Classes

Name	Description
TDAScript	Makes it possible to execute several SQL statements one by one.
TDAStatement	This class has attributes and methods for controlling single SQL statement of a script.
TDAStatements	Holds a collection of TDAStatement objects.

 Request Support DAC Forum Provide Feedback

5.8.1.1 TDAScript Class

Makes it possible to execute several SQL statements one by one.

For a list of all members of this type, see TDAScript members.

Unit

DAScript

Syntax

```
TDAScript = class(TComponent);
```

Remarks

Often it is necessary to execute several SQL statements one by one. This can be performed using a lot of components such as TCustomDASQL descendants. Usually it isn't the best solution. With only one TDAScript descedant component you can execute several SQL statements as one. This sequence of statements is called script. To separate single statements use semicolon (;) or slash (/) and for statements that can contain semicolon, only slash. Note that slash must be the first character in line.

Errors that occur during execution can be processed in the TDAScript.OnError event handler. By default, on error TDAScript shows exception and continues execution.

See Also

- TCustomDASQL

Request Support DAC Forum Provide Feedback

5.8.1.1.1 Members

TDAScript class overview.

Properties

Name	Description
Connection	Used to specify the connection in which the script will be executed.
DataSet	Refers to a dataset that holds the result set of query execution.
Debug	Used to display the script execution and all its parameter values.
Delimiter	Used to set the delimiter string that separates script statements.
EndLine	Used to get the current statement last line number in a script.
EndOffset	Used to get the offset in the

	last line of the current statement.
EndPos	Used to get the end position of the current statement.
Macros	Used to change SQL script text in design- or run-time easily.
SQL	Used to get or set script text.
StartLine	Used to get the current statement start line number in a script.
StartOffset	Used to get the offset in the first line of the current statement.
StartPos	Used to get the start position of the current statement in a script.
Statements	Contains a list of statements obtained from the SQL property.

Methods

Name	Description
BreakExec	Stops script execution.
ErrorOffset	Used to get the offset of the statement if the Execute method raised an exception.
Execute	Executes a script.
ExecuteFile	Executes SQL statements contained in a file.
ExecuteNext	Executes the next statement in the script and then stops.
ExecuteStream	Executes SQL statements contained in a stream object.
FindMacro	Finds a macro with the specified name.
MacroByName	Finds a macro with the specified name.

Events

Name	Description
AfterExecute	Occurs after a SQL script execution.
BeforeExecute	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError	Occurs when Oracle raises an error.

Request Support DAC Forum Provide Feedback

5.8.1.1.2 Properties

Properties of the **TDAScript** class.

For a complete list of the **TDAScript** class members, see the TDAScript Members topic.

Public

Name	Description
Connection	Used to specify the connection in which the script will be executed.
DataSet	Refers to a dataset that holds the result set of query execution.
EndLine	Used to get the current statement last line number in a script.
EndOffset	Used to get the offset in the last line of the current statement.
EndPos	Used to get the end position of the current statement.
StartLine	Used to get the current statement start line number in a script.
StartOffset	Used to get the offset in the first line of the current

	statement.
StartPos	Used to get the start position of the current statement in a script.
Statements	Contains a list of statements obtained from the SQL property.

Published

Name	Description
Debug	Used to display the script execution and all its parameter values.
Delimiter	Used to set the delimiter string that separates script statements.
Macros	Used to change SQL script text in design- or run-time easily.
SQL	Used to get or set script text.

See Also

- TDAScript Class
- TDAScript Class Members

Request Support DAC Forum Provide Feedback

5.8.1.1.2.1 Connection Property

Used to specify the connection in which the script will be executed.

Class

TDAScript

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify the connection in which the script will be executed. If Connection is not connected, the Execute method calls the Connect method of Connection.

Set at design-time by selecting from the list of provided TCustomDAConnection objects.

At run-time, set the Connection property to reference an existing TCustomDAConnection object.

See Also

- TCustomDAConnection

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.2 DataSet Property

Refers to a dataset that holds the result set of query execution.

Class

TDAScript

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

Set the DataSet property to assign a component that will be used by TOraScript to execute statements and to retrieve the results of the SELECT statements execution inside a script.

See Also

- ExecuteNext
- Execute

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.3 Debug Property

Used to display the script execution and all its parameter values.

Class

TDAScript

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

5.8.1.1.2.4 Delimiter Property

Used to set the delimiter string that separates script statements.

Class

TDAScript

Syntax

```
property Delimiter: string stored IsDelimiterStored;
```

Remarks

Use the Delimiter property to set the delimiter string that separates script statements. By default it is semicolon (;). You can use slash (/) to separate statements that can contain semicolon if the Delimiter property's default value is semicolon. Note that slash must be the first character in line.

5.8.1.1.2.5 EndLine Property

Used to get the current statement last line number in a script.

Class

TDAScript

Syntax

```
property EndLine: Int64;
```

Remarks

Use the EndLine property to get the current statement last line number in a script.

5.8.1.1.2.6 EndOffset Property

Used to get the offset in the last line of the current statement.

Class

TDAScript

Syntax

```
property EndOffset: Int64;
```

Remarks

Use the EndOffset property to get the offset in the last line of the current statement.

5.8.1.1.2.7 EndPos Property

Used to get the end position of the current statement.

Class

TDAScript

Syntax

```
property EndPos: Int64;
```

Remarks

Use the EndPos property to get the end position of the current statement (the position of the last character in the statement) in a script.

Request Support DAC Forum Provide Feedback

5.8.1.1.2.8 Macros Property

Used to change SQL script text in design- or run-time easily.

Class

TDAScript

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL script text in design- or run-time. Macros extend abilities of parameters and allow changing conditions in the WHERE clause or sort order in the ORDER BY clause. You just insert &MacroName in a SQL query text and change value of macro by the Macro property editor in design-time or the MacroByName function in run-time. In time of opening query macro is replaced by its value.

See Also

- TMacro

- MacroByName

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.9 SQL Property

Used to get or set script text.

Class

TDAScript

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to get or set script text.

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.10 StartLine Property

Used to get the current statement start line number in a script.

Class

TDAScript

Syntax

```
property StartLine: Int64;
```

Remarks

Use the StartLine property to get the current statement start line number in a script.

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.11 StartOffset Property

Used to get the offset in the first line of the current statement.

Class

TDAScript

Syntax

```
property StartOffset: Int64;
```

Remarks

Use the StartOffset property to get the offset in the first line of the current statement.

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.12 StartPos Property

Used to get the start position of the current statement in a script.

Class

TDAScript

Syntax

```
property StartPos: Int64;
```

Remarks

Use the StartPos property to get the start position of the current statement (the position of the first statement character) in a script.

 Request Support DAC Forum Provide Feedback

5.8.1.1.2.13 Statements Property

Contains a list of statements obtained from the SQL property.

Class

TDAScript

Syntax

```
property Statements: TDAStatements;
```

Remarks

Contains a list of statements that are obtained from the SQL property. Use the Access Statements property to view SQL statement, set parameters or execute the specified statement. Statements is a zero-based array of statement records. Index specifies the array element to access.

For example, consider the following script:

```
CREATE TABLE A (FIELD1 INTEGER);
INSERT INTO A VALUES(1);
INSERT INTO A VALUES(2);
INSERT INTO A VALUES(3);
CREATE TABLE B (FIELD1 INTEGER);
INSERT INTO B VALUES(1);
INSERT INTO B VALUES(2);
INSERT INTO B VALUES(3);
```

Note: The list of statements is created and filled when the value of Statements property is requested. That's why the first access to the Statements property can take a long time.

Example

You can use the Statements property in the following way:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i: integer;
begin
   with Script do
   begin
      for i := 0 to Statements.Count - 1 do
        if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
 Statements[i].Execute;
   end;
end;
```

See Also

- TDAStatements

Reserved.

5.8.1.1.3 Methods

Methods of the **TDAScript** class.

For a complete list of the **TDAScript** class members, see the TDAScript Members topic.

Public

Name	Description
BreakExec	Stops script execution.
ErrorOffset	Used to get the offset of the statement if the Execute method raised an exception.
Execute	Executes a script.
ExecuteFile	Executes SQL statements contained in a file.
ExecuteNext	Executes the next statement in the script and then stops.
ExecuteStream	Executes SQL statements contained in a stream object.
FindMacro	Finds a macro with the specified name.
MacroByName	Finds a macro with the specified name.

See Also

- TDAScript Class
- TDAScript Class Members

5.8.1.1.3.1 BreakExec Method

Stops script execution.

Class

TDAScript

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to stop script execution.

 Request Support DAC Forum Provide Feedback

5.8.1.1.3.2 ErrorOffset Method

Used to get the offset of the statement if the Execute method raised an exception.

Class

TDAScript

Syntax

```
function ErrorOffset: Int64;
```

Return Value

offset of an error.

Remarks

Call the ErrorOffset method to get the offset of the statement if the Execute method raised an exception.

See Also

- OnError

 Request Support DAC Forum Provide Feedback

5.8.1.1.3.3 Execute Method

Executes a script.

Class

TDAScript

Syntax

```
procedure Execute; virtual;
```

Remarks

Call the Execute method to execute a script. If Oracle raises an error, the OnError event occurs.

See Also

- ExecuteNext
- OnError
- ErrorOffset

 Request Support DAC Forum Provide Feedback

5.8.1.1.3.4 ExecuteFile Method

Executes SQL statements contained in a file.

Class

TDAScript

Syntax

```
procedure ExecuteFile(const FileName: string);
```

Parameters

FileName

Holds the file name.

Remarks

Call the ExecuteFile method to execute SQL statements contained in a file. Script doesn't load full content into memory. Reading and execution is performed by blocks of 64k size. Therefore, it is optimal to use it for big files.

 Request Support DAC Forum Provide Feedback

5.8.1.1.3.5 ExecuteNext Method

Executes the next statement in the script and then stops.

Class

TDAScript

Syntax

```
function ExecuteNext: boolean; virtual;
```

Return Value

True, if there are any statements left in the script, False otherwise.

Remarks

Use the ExecuteNext method to execute the next statement in the script statement and stop.

If Oracle raises an error, the OnError event occurs.

See Also

- Execute
- OnError
- ErrorOffset

5.8.1.1.3.6 ExecuteStream Method

Executes SQL statements contained in a stream object.

Class

TDAScript

Syntax

```
procedure ExecuteStream(Stream: TStream);
```

Parameters

Stream
Holds the stream object from which the statements will be executed.

Remarks

Call the ExecuteStream method to execute SQL statements contained in a stream object. Reading from the stream and execution is performed by blocks of 64k size.

Request Support DAC Forum Provide Feedback

5.8.1.1.3.7 FindMacro Method

Finds a macro with the specified name.

Class

TDAScript

Syntax

```
function FindMacro(Name: string): TMacro;
```

Parameters

Name
Holds the name of a macro to search for.

Return Value
TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro
- Macros
- MacroByName

Request Support DAC Forum Provide Feedback

5.8.1.1.3.8 MacroByName Method

Finds a macro with the specified name.

Class

TDAScript

Syntax

```
function MacroByName(Name: string): TMacro;
```

Parameters

Name

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro
- Macros
- FindMacro

5.8.1.1.4 Events

Events of the **TDAScript** class.

For a complete list of the **TDAScript** class members, see the TDAScript Members topic.

Published

Name	Description
AfterExecute	Occurs after a SQL script execution.
BeforeExecute	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError	Occurs when Oracle raises an error.

See Also

- TDAScript Class
- TDAScript Class Members

5.8.1.1.4.1 AfterExecute Event

Occurs after a SQL script execution.

Class

TDAScript

Syntax

```
property AfterExecute: TAfterStatementExecuteEvent;
```

Remarks

Occurs after a SQL script has been executed.

See Also

- Execute

 Request Support DAC Forum Provide Feedback

5.8.1.1.4.2 BeforeExecute Event

Occurs when taking a specific action before executing the current SQL statement is needed.

Class

TDAScript

Syntax

```
property BeforeExecute: TBeforeStatementExecuteEvent;
```

Remarks

Write the BeforeExecute event handler to take specific action before executing the current SQL statement. SQL holds text of the current SQL statement. Write SQL to change the statement that will be executed. Set Omit to True to skip statement execution.

 Request Support DAC Forum Provide Feedback

5.8.1.1.4.3 OnError Event

Occurs when Oracle raises an error.

Class

TDAScript

Syntax

```
property OnError: TOnErrorEvent;
```

Remarks

Occurs when Oracle raises an error.

Action indicates the action to take when the OnError handler exits. On entry into the handler, Action is always set to eaFail.

See Also

- ErrorOffset

5.8.1.2 TDAStatement Class

This class has attributes and methods for controlling single SQL statement of a script.

For a list of all members of this type, see TDAStatement members.

Unit

DAScript

Syntax

```
TDAStatement = class(TCollectionItem);
```

Remarks

TDAScript contains SQL statements, represented as TDAStatement objects. The TDAStatement class has attributes and methods for controlling single SQL statement of a script.

See Also

- TDAScript
- TDAStatements

5.8.1.2.1 Members

TDAStatement class overview.

Properties

Name	Description

EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.
EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.
Params	Contains parasmeters for an SQL statement.
Script	Used to determine the TDAScript object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

Methods

Name	Description
Execute	Executes a statement.

Request Support DAC Forum Provide Feedback

5.8.1.2.2 Properties

Properties of the **TDAStatement** class.

For a complete list of the **TDAStatement** class members, see the TDAStatement Members topic.

Public

Name	Description

EndLine	Used to determine the number of the last statement line in a script.
EndOffset	Used to get the offset in the last line of the statement.
EndPos	Used to get the end position of the statement in a script.
Omit	Used to avoid execution of a statement.
Params	Contains parasmeters for an SQL statement.
Script	Used to determine the TDAScript object the SQL Statement belongs to.
SQL	Used to get or set the text of an SQL statement.
StartLine	Used to determine the number of the first statement line in a script.
StartOffset	Used to get the offset in the first line of a statement.
StartPos	Used to get the start position of the statement in a script.

See Also

- TDAStatement Class
- TDAStatement Class Members

Request Support DAC Forum Provide Feedback

5.8.1.2.2.1 EndLine Property

Used to determine the number of the last statement line in a script.

Class

TDAStatement

Syntax

```
property EndLine: integer;
```

Remarks

Use the EndLine property to determine the number of the last statement line in a script.

Request Support DAC Forum Provide Feedback

5.8.1.2.2.2 EndOffset Property

Used to get the offset in the last line of the statement.

Class

TDAStatement

Syntax

```
property EndOffset: integer;
```

Remarks

Use the EndOffset property to get the offset in the last line of the statement.

Request Support DAC Forum Provide Feedback

5.8.1.2.2.3 EndPos Property

Used to get the end position of the statement in a script.

Class

TDAStatement

Syntax

```
property EndPos: integer;
```

Remarks

Use the EndPos property to get the end position of the statement (the position of the last character in the statement) in a script.

Request Support DAC Forum Provide Feedback

Reserved.

5.8.1.2.2.4 Omit Property

Used to avoid execution of a statement.

Class

TDAStatement

Syntax

```
property Omit: boolean;
```

Remarks

Set the Omit property to True to avoid execution of a statement.

 Request Support DAC Forum Provide Feedback

5.8.1.2.2.5 Params Property

Contains parasmeters for an SQL statement.

Class

TDAStatement

Syntax

```
property Params: TDAParams;
```

Remarks

Contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

See Also

- TDAParam

 Request Support DAC Forum Provide Feedback

5.8.1.2.2.6 Script Property

Used to determine the TDAScript object the SQL Statement belongs to.

Class

TDAStatement

Syntax

```
property Script: TDAScript;
```

Remarks

Use the Script property to determine the TDAScript object the SQL Statement belongs to.

 Request Support DAC Forum Provide Feedback

5.8.1.2.2.7 SQL Property

Used to get or set the text of an SQL statement.

Class

TDAStatement

Syntax

```
property SQL: string;
```

Remarks

Use the SQL property to get or set the text of an SQL statement.

 Request Support DAC Forum Provide Feedback

5.8.1.2.2.8 StartLine Property

Used to determine the number of the first statement line in a script.

Class

TDAStatement

Syntax

```
property StartLine: integer;
```

Remarks

Use the StartLine property to determine the number of the first statement line in a script.

5.8.1.2.2.9 StartOffset Property

Used to get the offset in the first line of a statement.

Class

TDAStatement

Syntax

```
property StartOffset: integer;
```

Remarks

Use the StartOffset property to get the offset in the first line of a statement.

5.8.1.2.2.10 StartPos Property

Used to get the start position of the statement in a script.

Class

TDAStatement

Syntax

```
property StartPos: integer;
```

Remarks

Use the StartPos property to get the start position of the statement (the position of the first statement character) in a script.

 Request Support DAC Forum Provide Feedback

5.8.1.2.3 Methods

Methods of the **TDAStatement** class.

For a complete list of the **TDAStatement** class members, see the TDAStatement Members topic.

Public

Name	Description
Execute	Executes a statement.

See Also

- TDAStatement Class
- TDAStatement Class Members

 Request Support DAC Forum Provide Feedback

5.8.1.2.3.1 Execute Method

Executes a statement.

Class

TDAStatement

Syntax

```
procedure Execute;
```

Remarks

Use the Execute method to execute a statement.

 Request Support DAC Forum Provide Feedback

5.8.1.3 TDAStatements Class

Holds a collection of TDAStatement objects.

For a list of all members of this type, see TDAStatements members.

Unit

DAScript

Syntax

```
TDAStatements = class(TCollection);
```

Remarks

Each TDAStatements holds a collection of TDAStatement objects. TDAStatements maintains an index of the statements in its Items array. The Count property contains the number of statements in the collection. Use TDAStatements class to manipulate script SQL statements.

See Also

- TDAScript
- TDAStatement

 Request Support DAC Forum Provide Feedback

5.8.1.3.1 Members

TDAStatements class overview.

Properties

Name	Description
Items	Used to access separate script statements.

Request Support DAC Forum Provide Feedback

5.8.1.3.2 Properties

Properties of the **TDAStatements** class.

For a complete list of the **TDAStatements** class members, see the TDAStatements Members topic.

Public

Name	Description
Items	Used to access separate script statements.

See Also

- TDAStatements Class
- TDAStatements Class Members

Request Support DAC Forum Provide Feedback

5.8.1.3.2.1 Items Property(Indexer)

Used to access separate script statements.

Class

TDAStatements

Syntax

```
property Items[Index: Integer]: TDAStatement; default;
```

Parameters

Index

Holds the index value.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of TDAStatement.

See Also

- TDAStatement

 Request Support DAC Forum Provide Feedback

5.8.2 Types

Types in the **DAScript** unit.

Types

Name	Description
TAfterStatementExecuteEvent	This type is used for the TDAScript.AfterExecute event.
TBeforeStatementExecuteEvent	This type is used for the TDAScript.BeforeExecute event.
TOnErrorEvent	This type is used for the TDAScript.OnError event.

 Request Support DAC Forum Provide Feedback

5.8.2.1 TAfterStatementExecuteEvent Procedure Reference

This type is used for the TDAScript.AfterExecute event.

Unit

DAScript

Syntax

```
TAfterStatementExecuteEvent = procedure (Sender: TObject; SQL:
string) of object;
```

Parameters

Sender
An object that raised the event.

SQL
Holds the passed SQL statement.

Request Support DAC Forum Provide Feedback

5.8.2.2 TBeforeStatementExecuteEvent Procedure Reference

This type is used for the TDAScript.BeforeExecute event.

Unit

DAScript

Syntax

```
TBeforeStatementExecuteEvent = procedure (Sender: TObject; var
SQL: string; var Omit: boolean) of object;
```

Parameters

Sender
An object that raised the event.

SQL
Holds the passed SQL statement.

Omit
True, if the statement execution should be skipped.

Request Support DAC Forum Provide Feedback

5.8.2.3 TOnErrorEvent Procedure Reference

This type is used for the TDAScript.OnError event.

Unit

DAScript

Syntax

```
TOnErrorEvent = procedure (Sender: TObject; E: Exception; SQL:
string; var Action: TErrorAction) of object;
```

Parameters

Sender

An object that raised the event.

E

The error code.

SQL

Holds the passed SQL statement.

Action

The action to take when the OnError handler exits.

 Request Support DAC Forum Provide Feedback

5.8.3 Enumerations

Enumerations in the **DAScript** unit.

Enumerations

Name	Description
TErrorAction	Indicates the action to take when the OnError handler exits.

 Request Support DAC Forum Provide Feedback

5.8.3.1 TErrorAction Enumeration

Indicates the action to take when the OnError handler exits.

Unit

DAScript

Syntax

```
TErrorAction = (eaAbort, eaFail, eaException, eaContinue);
```

Values

Value	Meaning
eaAbort	Abort execution without displaying an error message.
eaContinue	Continue execution.
eaException	In Delphi 6 and higher exception is handled by the Application.HandleException method.
eaFail	Abort execution and display an error message.

Request Support DAC Forum Provide Feedback

5.9 DASQLMonitor

This unit contains the base class for the TOraSQLMonitor component.

Classes

Name	Description
TCustomDASQLMonitor	A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.
TDBMonitorOptions	This class holds options for dbMonitor.

Types

Name	Description
TDATraceFlags	Represents the set of TDATraceFlag.
TMonitorOptions	Represents the set of TMonitorOption.
TOnSQLEvent	This type is used for the TCustomDASQLMonitor.OnSQL event.

Enumerations

Name	Description
TDATraceFlag	Use TraceFlags to specify which database operations

	the monitor should track in an application at runtime.
TMonitorOption	Used to define where information from SQLMonitor will be dispalyed.

 Request Support DAC Forum Provide Feedback

5.9.1 Classes

Classes in the **DASQLMonitor** unit.

Classes

Name	Description
TCustomDASQLMonitor	A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.
TDBMonitorOptions	This class holds options for dbMonitor.

 Request Support DAC Forum Provide Feedback

5.9.1.1 TCustomDASQLMonitor Class

A base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively.

For a list of all members of this type, see TCustomDASQLMonitor members.

Unit

DASQLMonitor

Syntax

```
TCustomDASQLMonitor = class(TComponent);
```

Remarks

TCustomDASQLMonitor is a base class that introduces properties and methods to monitor dynamic SQL execution in database applications interactively. TCustomDASQLMonitor provides two ways of displaying debug information. It monitors either by dialog window or by Borland's proprietary SQL Monitor. Furthermore to receive debug information use the TCustomDASQLMonitor.OnSQL event.

In applications use descendants of TCustomDASQLMonitor.

Request Support DAC Forum Provide Feedback

5.9.1.1.1 Members

TCustomDASQLMonitor class overview.

Properties

Name	Description
Active	Used to activate monitoring of SQL.
DBMonitorOptions	Used to set options for dbMonitor.
Options	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags	Used to specify which database operations the monitor should track in an application at runtime.

Events

Name	Description
OnSQL	Occurs when tracing of SQL activity on database components is needed.

Request Support DAC Forum Provide Feedback

5.9.1.1.2 Properties

Properties of the **TCustomDASQLMonitor** class.

For a complete list of the **TCustomDASQLMonitor** class members, see the TCustomDASQLMonitor Members topic.

Public

Name	Description
Active	Used to activate monitoring of SQL.
DBMonitorOptions	Used to set options for dbMonitor.
Options	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags	Used to specify which database operations the monitor should track in an application at runtime.

See Also

- TCustomDASQLMonitor Class
- TCustomDASQLMonitor Class Members

Request Support DAC Forum Provide Feedback

5.9.1.1.2.1 Active Property

Used to activate monitoring of SQL.

Class

TCustomDASQLMonitor

Syntax

```
property Active: boolean default True;
```

Remarks

Set the Active property to True to activate monitoring of SQL.

See Also

- OnSQL

 Request Support DAC Forum Provide Feedback

5.9.1.1.2.2 DBMonitorOptions Property

Used to set options for dbMonitor.

Class

TCustomDASQLMonitor

Syntax

```
property DBMonitorOptions: TDBMonitorOptions;
```

Remarks

Use DBMonitorOptions to set options for dbMonitor.

 Request Support DAC Forum Provide Feedback

5.9.1.1.2.3 Options Property

Used to include the desired properties for TCustomDASQLMonitor.

Class

TCustomDASQLMonitor

Syntax

```
property Options: TMonitorOptions default [moDialog,
moSQLMonitor, moDBMonitor, moCustom];
```

Remarks

Set Options to include the desired properties for TCustomDASQLMonitor.

See Also

- OnSQL

5.9.1.1.2.4 TraceFlags Property

Used to specify which database operations the monitor should track in an application at runtime.

Class

TCustomDASQLMonitor

Syntax

```
property TraceFlags: TDATraceFlags default [tfQPrepare,
tfQExecute, tfError, tfConnect, tfTransact, tfParams, tfMisc];
```

Remarks

Use the TraceFlags property to specify which database operations the monitor should track in an application at runtime.

See Also

- OnSQL

5.9.1.1.3 Events

Events of the **TCustomDASQLMonitor** class.

For a complete list of the **TCustomDASQLMonitor** class members, see the TCustomDASQLMonitor Members topic.

Public

Name	Description

OnSQL	Occurs when tracing of SQL activity on database components is needed.

See Also

- TCustomDASQLMonitor Class
- TCustomDASQLMonitor Class Members

 Request Support DAC Forum Provide Feedback

5.9.1.1.3.1 OnSQL Event

Occurs when tracing of SQL activity on database components is needed.

Class

TCustomDASQLMonitor

Syntax

```
property OnSQL: TOnSQLEvent;
```

Remarks

Write the OnSQL event handler to let an application trace SQL activity on database components. The Text parameter holds the detected SQL statement. Use the Flag parameter to make selective processing of SQL in the handler body.

See Also

- TraceFlags

 Request Support DAC Forum Provide Feedback

5.9.1.2 TDBMonitorOptions Class

This class holds options for dbMonitor.

For a list of all members of this type, see TDBMonitorOptions members.

Unit

DASQLMonitor

Syntax

```
TDBMonitorOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.9.1.2.1 Members

TDBMonitorOptions class overview.

Properties

Name	Description
Host	Used to set the host name or IP address of the computer where dbMonitor application runs.
Port	Used to set the port number for connecting to dbMonitor.
ReconnectTimeout	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
SendTimeout	Used to set timeout for sending events to dbMonitor.

 Request Support DAC Forum Provide Feedback

5.9.1.2.2 Properties

Properties of the **TDBMonitorOptions** class.

For a complete list of the **TDBMonitorOptions** class members, see the TDBMonitorOptions Members topic.

Published

Name	Description
Host	Used to set the host name or IP address of the computer where dbMonitor application runs.
Port	Used to set the port number for connecting to dbMonitor.
ReconnectTimeout	Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.
SendTimeout	Used to set timeout for sending events to dbMonitor.

See Also

- TDBMonitorOptions Class
- TDBMonitorOptions Class Members

Request Support DAC Forum Provide Feedback

5.9.1.2.2.1 Host Property

Used to set the host name or IP address of the computer where dbMonitor application runs.

Class

TDBMonitorOptions

Syntax

```
property Host: string;
```

Remarks

Use the Host property to set the host name or IP address of the computer where dbMonitor application runs.

dbMonitor supports remote monitoring. You can run dbMonitor on a different computer than monitored application runs. In this case you need to set the Host property to the corresponding computer name.

 Request Support DAC Forum Provide Feedback

5.9.1.2.2.2 Port Property

Used to set the port number for connecting to dbMonitor.

Class

TDBMonitorOptions

Syntax

```
property Port: integer default DBMonitorPort;
```

Remarks

Use the Port property to set the port number for connecting to dbMonitor.

 Request Support DAC Forum Provide Feedback

5.9.1.2.2.3 ReconnectTimeout Property

Used to set the minimum time that should be spent before reconnecting to dbMonitor is allowed.

Class

TDBMonitorOptions

Syntax

```
property ReconnectTimeout: integer default
DefaultReconnectTimeout;
```

Remarks

Use the ReconnectTimeout property to set the minimum time (in milliseconds) that should be spent before allowing reconnecting to dbMonitor. If an error occurs when the component sends an event to dbMonitor (dbMonitor is not running), next events are ignored and the component does not restore the connection until ReconnectTimeout is over.

 Request Support DAC Forum Provide Feedback

5.9.1.2.2.4 SendTimeout Property

Used to set timeout for sending events to dbMonitor.

Class

TDBMonitorOptions

Syntax

```
property SendTimeout: integer default DefaultSendTimeout;
```

Remarks

Use the SendTimeout property to set timeout (in milliseconds) for sending events to dbMonitor. If dbMonitor does not respond in the specified timeout, event is ignored.

Request Support DAC Forum Provide Feedback

5.9.2 Types

Types in the **DASQLMonitor** unit.

Types

Name	Description
TDATraceFlags	Represents the set of TDATraceFlag.
TMonitorOptions	Represents the set of TMonitorOption.
TOnSQLEvent	This type is used for the TCustomDASQLMonitor.OnSQL event.

Request Support DAC Forum Provide Feedback

5.9.2.1 TDATraceFlags Set

Represents the set of TDATraceFlag.

Unit

DASQLMonitor

Syntax

```
TDATraceFlags = set of TDATraceFlag;
```


Request Support DAC Forum Provide Feedback

5.9.2.2 TMonitorOptions Set

Represents the set of TMonitorOption.

Unit

DASQLMonitor

Syntax

```
TMonitorOptions = set of TMonitorOption;
```


Request Support DAC Forum Provide Feedback

5.9.2.3 TOnSQLEvent Procedure Reference

This type is used for the TCustomDASQLMonitor.OnSQL event.

Unit

DASQLMonitor

Syntax

```
TOnSQLEvent = procedure (Sender: TObject; Text: string; Flag:
TDATraceFlag) of object;
```

Parameters

Sender

An object that raised the event.

Text

Holds the detected SQL statement.

Flag

Use the Flag parameter to make selective processing of SQL in the handler body.

 Request Support DAC Forum Provide Feedback

5.9.3 Enumerations

Enumerations in the **DASQLMonitor** unit.

Enumerations

Name	Description
TDATraceFlag	Use TraceFlags to specify which database operations the monitor should track in an application at runtime.
TMonitorOption	Used to define where information from SQLMonitor will be dispalyed.

 Request Support DAC Forum Provide Feedback

5.9.3.1 TDATraceFlag Enumeration

Use TraceFlags to specify which database operations the monitor should track in an application at runtime.

Unit

DASQLMonitor

Syntax

```
TDATraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt,
tfConnect, tfTransact, tfBlob, tfService, tfMisc, tfParams,
tfObjDestroy, tfPool);
```

Values

Value	Meaning
tfBlob	This option is declared for future use.
tfConnect	Establishing a connection.
tfError	Errors of query execution.
tfMisc	This option is declared for future use.
tfObjDestroy	Destroying of components.
tfParams	Representing parameter values for tfQPrepare and tfQExecute.
tfPool	Connection pool operations.
tfQExecute	Execution of the queries.
tfQFetch	This option is declared for future use.
tfQPrepare	Queries preparation.
tfService	This option is declared for future use.
tfStmt	This option is declared for future use.
tfTransact	Processing transactions.

Request Support DAC Forum Provide Feedback

5.9.3.2 TMonitorOption Enumeration

Used to define where information from SQLMonitor will be dispalyed.

Unit

DASQLMonitor

Syntax

```
TMonitorOption = (moDialog, moSQLMonitor, moDBMonitor, moCustom,
moHandled);
```

Values

Value	Meaning
moCustom	Monitoring of SQL for individual components is allowed. Set Debug properties in SQL-related components to True to let TCustomDASQLMonitor instance to monitor their behavior. Has effect when moDialog is included.
moDBMonitor	Debug information is displayed in DBMonitor.

moDialog	Debug information is displayed in debug window.
moHandled	Component handle is included into the event description string.
moSQLMonitor	Debug information is displayed in Borland SQL Monitor.

Request Support DAC Forum Provide Feedback

5.10 DBAccess

This unit contains base classes for most of the components.

Classes

Name	Description
EDAError	A base class for exceptions that are raised when an error occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
TDACondition	Represents a condition from the TDAConditions list.

TDAConditions	Holds a collection of TDACondition objects.
TDAConnectionOptions	This class allows setting up the behaviour of the TDAConnection class.
TDAConnectionSSLOptions	This class is used to set up the SSL options.
TDADataSetOptions	This class allows setting up the behaviour of the TDADataSet class.
TDAEncryption	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that formes rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to represent the values of the parameters set.
TDAParams	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
TDATransaction	A base class that implements functionality for controlling transactions.
TMacro	Object that represents the value of a macro.
TMacros	Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.
TPoolingOptions	This class allows setting up the behaviour of the connection pool.

TSmartFetchOptions	Smart fetch options are used to set up the behavior of the SmartFetch mode.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.
TAfterFetchEvent	This type is used for the TCustomDADataSet.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataSet.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption.
TUpdateExecuteEvent	This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

Enumerations

Name	Description
TCheckMode	Specifies the action to take when another user makes modifications to a record.
TLabelSet	Sets the languauge of labels in the connect dialog.

TLockMode	Specifies the lock mode.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

Variables

Name	Description
BaseSQLOldBehavior	After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.
SQLGeneratorCompatibility	The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

Request Support DAC Forum Provide Feedback

5.10.1 Classes

Classes in the **DBAccess** unit.

Classes

Name	Description
EDAError	A base class for exceptions that are raised when an error

	occurs on the server side.
TCRDataSource	Provides an interface between a DAC dataset components and data-aware controls on a form.
TCustomConnectDialog	A base class for the connect dialog components.
TCustomDAConnection	A base class for components used to establish connections.
TCustomDADataSet	Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.
TCustomDASQL	A base class for components executing SQL statements that do not return result sets.
TCustomDAUpdateSQL	A base class for components that provide DML statements for more flexible control over data modifications.
TDACondition	Represents a condition from the TDAConditions list.
TDAConditions	Holds a collection of TDACondition objects.
TDAConnectionOptions	This class allows setting up the behaviour of the TDAConnection class.
TDAConnectionSSLOptions	This class is used to set up the SSL options.
TDADataSetOptions	This class allows setting up the behaviour of the TDADataSet class.
TDAEncryption	Used to specify the options of the data encryption in a dataset.
TDAMapRule	Class that formes rules for Data Type Mapping.
TDAMapRules	Used for adding rules for DataSet fields mapping with both identifying by field

	name and by field type and Delphi field types.
TDAMetaData	A class for retrieving metainformation of the specified database objects in the form of dataset.
TDAParam	A class that forms objects to represent the values of the parameters set.
TDAParams	This class is used to manage a list of TDAParam objects for an object that uses field parameters.
TDATransaction	A base class that implements functionality for controlling transactions.
TMacro	Object that represents the value of a macro.
TMacros	Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.
TPoolingOptions	This class allows setting up the behaviour of the connection pool.
TSmartFetchOptions	Smart fetch options are used to set up the behavior of the SmartFetch mode.

Request Support DAC Forum Provide Feedback

5.10.1.1 EDAError Class

A base class for exceptions that are raised when an error occurs on the server side.

For a list of all members of this type, see EDAError members.

Unit

DBAccess

Syntax

```
EDAError = class(EDatabaseError);
```

Remarks

EDAError is a base class for exceptions that are raised when an error occurs on the server side.

Request Support DAC Forum Provide Feedback

5.10.1.1.1 Members

EDAError class overview.

Properties

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

© 1997-2024 Devart. All Rights Reserved.
Request Support DAC Forum Provide Feedback

5.10.1.1.2 Properties

Properties of the **EDAError** class.

For a complete list of the **EDAError** class members, see the EDAError Members topic.

Public

Name	Description
Component	Contains the component that caused the error.
ErrorCode	Determines the error code returned by the server.

See Also

- EDAError Class

- EDAError Class Members

5.10.1.1.2.1 Component Property

Contains the component that caused the error.

Class

EDAError

Syntax

```
property Component: TObject;
```

Remarks

The Component property contains the component that caused the error.

5.10.1.1.2.2 ErrorCode Property

Determines the error code returned by the server.

Class

EDAError

Syntax

```
property ErrorCode: integer;
```

Remarks

Use the ErrorCode property to determine the error code returned by Oracle. This value is always positive.

See Also

- TOraErrorHandler.OnError

- TOraErrorHandler.OnError

5.10.1.2 TCRDataSource Class

Provides an interface between a DAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TCRDataSource members.

Unit

DBAccess

Syntax

```
TCRDataSource = class(TDataSource);
```

Remarks

TCRDataSource provides an interface between a DAC dataset components and data-aware controls on a form.

TCRDataSource inherits its functionality directly from the TDataSource component.

At design time assign individual data-aware components' DataSource properties from their drop-down listboxes.

5.10.1.2.1 Members

TCRDataSource class overview.

5.10.1.3 TCustomConnectDialog Class

A base class for the connect dialog components.

For a list of all members of this type, see TCustomConnectDialog members.

Unit

DBAccess

Syntax

```
TCustomConnectDialog = class(TComponent);
```

Remarks

TCustomConnectDialog is a base class for the connect dialog components. It provides functionality to show a dialog box where user can edit username, password and server name before connecting to a database. You can customize captions of buttons and labels by their properties.

5.10.1.3.1 Members

TCustomConnectDialog class overview.

Properties

Name	Description
CancelButton	Used to specify the label for the Cancel button.
Caption	Used to set the caption of dialog box.
ConnectButton	Used to specify the label for the Connect button.
DialogClass	Used to specify the class of the form that will be displayed to enter login information.
LabelSet	Used to set the language of buttons and labels captions.
PasswordLabel	Used to specify a prompt for password edit.
Retries	Used to indicate the number of retries of failed

	connections.
SavePassword	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel	Used to specify a prompt for the server name edit.
StoreLogInfo	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel	Used to specify a prompt for username edit.

Methods

Name	Description
Execute	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList	Retrieves a list of available server names.

Request Support DAC Forum Provide Feedback

5.10.1.3.2 Properties

Properties of the **TCustomConnectDialog** class.

For a complete list of the **TCustomConnectDialog** class members, see the TCustomConnectDialog Members topic.

Public

Name	Description
CancelButton	Used to specify the label for the Cancel button.
Caption	Used to set the caption of dialog box.
ConnectButton	Used to specify the label for the Connect button.

DialogClass	Used to specify the class of the form that will be displayed to enter login information.
LabelSet	Used to set the language of buttons and labels captions.
PasswordLabel	Used to specify a prompt for password edit.
Retries	Used to indicate the number of retries of failed connections.
SavePassword	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel	Used to specify a prompt for the server name edit.
StoreLogInfo	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel	Used to specify a prompt for username edit.

See Also

- TCustomConnectDialog Class
- TCustomConnectDialog Class Members

5.10.1.3.2.1 CancelButton Property

Used to specify the label for the Cancel button.

Class

TCustomConnectDialog

Syntax

```
property CancelButton: string;
```

Remarks

Use the CancelButton property to specify the label for the Cancel button.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.2 Caption Property

Used to set the caption of dialog box.

Class

TCustomConnectDialog

Syntax

```
property Caption: string;
```

Remarks

Use the Caption property to set the caption of dialog box.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.3 ConnectButton Property

Used to specify the label for the Connect button.

Class

TCustomConnectDialog

Syntax

```
property ConnectButton: string;
```

Remarks

Use the ConnectButton property to specify the label for the Connect button.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.4 DialogClass Property

Used to specify the class of the form that will be displayed to enter login information.

Class

TCustomConnectDialog

Syntax

```
property DialogClass: string;
```

Remarks

Use the DialogClass property to specify the class of the form that will be displayed to enter login information. When this property is blank, TCustomConnectDialog uses the default form - TConnectForm. You can write your own login form to enter login information and assign its class name to the DialogClass property. Each login form must have ConnectDialog: TCustomConnectDialog published property to access connection information. For details see the implementation of the connect form which sources are in the Lib subdirectory of the ODAC installation directory.

See Also

- GetServerList

Request Support DAC Forum Provide Feedback

5.10.1.3.2.5 LabelSet Property

Used to set the language of buttons and labels captions.

Class

TCustomConnectDialog

Syntax

```
property LabelSet: TLabelSet default lsEnglish;
```

Remarks

Use the LabelSet property to set the language of labels and buttons captions.

The default value is IsEnglish.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.6 PasswordLabel Property

Used to specify a prompt for password edit.

Class

TCustomConnectDialog

Syntax

```
property PasswordLabel: string;
```

Remarks

Use the PasswordLabel property to specify a prompt for password edit.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.7 Retries Property

Used to indicate the number of retries of failed connections.

Class

TCustomConnectDialog

Syntax

```
property Retries: word default 3;
```

Remarks

Use the Retries property to determine the number of retries of failed connections.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.8 SavePassw ord Property

Used for the password to be displayed in ConnectDialog in asterisks.

Class

TCustomConnectDialog

Syntax

```
property SavePassword: boolean default False;
```

Remarks

If True, and the Password property of the connection instance is assigned, the password in ConnectDialog is displayed in asterisks.

 Request Support DAC Forum Provide Feedback

5.10.1.3.2.9 ServerLabel Property

Used to specify a prompt for the server name edit.

Class

TCustomConnectDialog

Syntax

```
property ServerLabel: string;
```

Remarks

Use the ServerLabel property to specify a prompt for the server name edit.

 Request Support DAC Forum Provide Feedback

5.10.1.3.2.10 StoreLogInfo Property

Used to specify whether the login information should be kept in system registry after a connection was established.

Class

TCustomConnectDialog

Syntax

```
property StoreLogInfo: boolean default True;
```

Remarks

Use the StoreLogInfo property to specify whether to keep login information in system registry after a connection was established using provided username, password and servername.

Set this property to True to store login information.

The default value is True.

Request Support DAC Forum Provide Feedback

5.10.1.3.2.11 UsernameLabel Property

Used to specify a prompt for username edit.

Class

TCustomConnectDialog

Syntax

```
property UsernameLabel: string;
```

Remarks

Use the UsernameLabel property to specify a prompt for username edit.

Request Support DAC Forum Provide Feedback

5.10.1.3.3 Methods

Methods of the **TCustomConnectDialog** class.

For a complete list of the **TCustomConnectDialog** class members, see the

TCustomConnectDialog Members topic.

Public

Name	Description
Execute	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList	Retrieves a list of available server names.

See Also

- TCustomConnectDialog Class
- TCustomConnectDialog Class Members

Request Support DAC Forum Provide Feedback

5.10.1.3.3.1 Execute Method

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.

Class

TCustomConnectDialog

Syntax

```
function Execute: boolean; virtual;
```

Return Value

True, if connected.

Remarks

Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button. Returns True if connected. If user clicks Cancel, Execute returns False.

In the case of failed connection Execute offers to connect repeat Retries times.

Request Support DAC Forum Provide Feedback

5.10.1.3.3.2 GetServerList Method

Retrieves a list of available server names.

Class

TCustomConnectDialog

Syntax

```
procedure GetServerList(List: TStrings); virtual;
```

Parameters

List

Holds a list of available server names.

Remarks

Call the GetServerList method to retrieve a list of available server names. It is particularly relevant for writing custom login form.

See Also

- DialogClass

5.10.1.4 TCustomDAConnection Class

A base class for components used to establish connections.

For a list of all members of this type, see TCustomDAConnection members.

Unit

DBAccess

Syntax

```
TCustomDAConnection = class(TCustomConnection);
```

Remarks

TCustomDAConnection is a base class for components that establish connection with database, provide customised login support, and perform transaction control.

Do not create instances of TCustomDAConnection. To add a component that represents a connection to a source of data, use descendants of the TCustomDAConnection class.

 Request Support DAC Forum Provide Feedback

5.10.1.4.1 Members

TCustomDAConnection class overview.

Properties

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.
InTransaction	Indicates whether the transaction is active.
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server

	name for login.
Username	Used to supply a user name for login.

Methods

Name	Description
ApplyUpdates	Overloaded. Applies changes in datasets.
Commit	Commits current transaction.
Connect	Establishes a connection to the server.
CreateSQL	Creates a component for queries execution.
Disconnect	Performs disconnect.
ExecProc	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx	Allows to execute a stored procedure or function.
ExecSQL	Executes a SQL statement with parameters.
ExecSQLEx	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames	Returns a database list from the server.
GetKeyFieldNames	Provides a list of available key field names.
GetStoredProcNames	Returns a list of stored procedures from the server.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.
Ping	Used to check state of connection to the server.

RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.
Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

Events

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

Request Support DAC Forum Provide Feedback

5.10.1.4.2 Properties

Properties of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the TCustomDAConnection Members topic.

Public

Name	Description
ConnectDialog	Allows to link a TCustomConnectDialog component.
ConnectString	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL	Allows customizing line breaks in string fields and parameters.
InTransaction	Indicates whether the transaction is active.

LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Options	Specifies the connection behavior.
Password	Serves to supply a password for login.
Pooling	Enables or disables using connection pool.
PoolingOptions	Specifies the behaviour of connection pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.

See Also

- TCustomDAConnection Class
- TCustomDAConnection Class Members

5.10.1.4.2.1 ConnectDialog Property

Allows to link a TCustomConnectDialog component.

Class

TCustomDAConnection

Syntax

```
property ConnectDialog: TCustomConnectDialog;
```

Remarks

Use the ConnectDialog property to assign to connection a TCustomConnectDialog component.

See Also

- TCustomConnectDialog

5.10.1.4.2.2 ConnectString Property

Used to specify the connection information, such as: UserName, Password, Server, etc.

Class

TCustomDAConnection

Syntax

```
property ConnectString: string stored False;
```

Remarks

ODAC recognizes an ODBC-like syntax in provider string property values. Within the string, elements are delimited by using a semicolon. Each element consists of a keyword, an equal sign character, and the value passed on initialization. For example:

```
Server=London1;User ID=nancyd
```

Connection parameters

The following connection parameters can be used to customize connection:

Parameter Name	Description
LoginPrompt	Specifies whether a login dialog appears immediately before opening a new connection.
Pooling	Enables or disables using connection pool.
ConnectionLifeTime	Used to specify the maximum time during which an opened connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in connection pool.
Validate Connection	Used for a connection to be validated when

	it is returned from the pool.
Server	Serves to supply the server name for login.
Username	Used to supply a user name for login.
Password	Used to supply a user name for login.
Charset	Used to set the character set that ODAC uses to read and write character data.
UseUnicode	Used to enable or disable Unicode support.
Port	Used to specify the port number for the connection. Available in Direct Mode only.
ConnectionTimeout	Used to specify the amount of time before an attempt to make a connection is considered unsuccessful.
Direct	Used for ODAC to connect directly over TCP/IP (in Direct mode) and without requiring Oracle software on the client side.
IPVersion	Used to specify the version of the Internet Protocol.
SID	Used to specify the security identifier.
ServiceName	Used to specify the name of the service.
ConnectMode	Used to specify the system privileges to use when a user connects to the server.
HomeName	Used to select the Oracle client to use with the application.
Schema	Used to change the current schema of the session to the specified schema.

Old-style conenction string

The old connection string format is also supported:

```
user_name/passwd@database
```

where: user_name - username for logging in to the server; passwd - password to log in; database - the database alias from the tnsnames.ora file.

EZCONNECT connection string

ODAC also supports Oracle's easy connect naming method . An example of EZCONNECT connection string:

```
username/password@host:port/service_name
```

See Also

- Password
- Username
- Server
- Connect

 Request Support DAC Forum Provide Feedback

5.10.1.4.2.3 ConvertEOL Property

Allows customizing line breaks in string fields and parameters.

Class

TCustomDAConnection

Syntax

```
property ConvertEOL: boolean default False;
```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including the CLOB and LONG fields) with ConvertEOL = True, dataset converts their line breaks from the LF to CRLF form. And when posting strings to server with ConvertEOL turned on, their line breaks are converted from CRLF to LF form. By default, strings are not converted.

 Request Support DAC Forum Provide Feedback

5.10.1.4.2.4 InTransaction Property

Indicates whether the transaction is active.

Class

TCustomDAConnection

Syntax

```
property InTransaction: boolean;
```

Remarks

Examine the InTransaction property at runtime to determine whether user transaction is currently in progress. In other words InTransaction is set to True when user explicitly calls StartTransaction. Calling Commit or Rollback sets InTransaction to False. The value of the InTransaction property cannot be changed directly.

Important note: The InTransaction property always shows actual user transaction state on the server. This means that if transaction was implicitly ended by server-side logic, InTransaction becomes False.

See Also

- StartTransaction
- Commit
- Rollback

5.10.1.4.2.5 LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Class

TCustomDAConnection

Syntax

```
property LoginPrompt default DefValLoginPrompt;
```

Remarks

Specifies whether a login dialog appears immediately before opening a new connection. If ConnectDialog is not specified, the default connect dialog will be shown. The connect dialog will appear only if the OdacVcl unit appears to the uses clause.

5.10.1.4.2.6 Options Property

Specifies the connection behavior.

Class

TCustomDAConnection

Syntax

```
property Options: TDAConnectionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of the connection.

Descriptions of all options are in the table below.

Option Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

See Also

- Disconnected Mode
- Working in an Unstable Network

5.10.1.4.2.7 Password Property

Serves to supply a password for login.

Class

TCustomDAConnection

Syntax

```
property Password: string stored False;
```

Remarks

Use the Password property to supply a password to handle server's request for a login. Application server can use the TOraSession.ProxySession property and verify user password itself.

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- Username
- Server
- TOraSession.ProxySession

5.10.1.4.2.8 Pooling Property

Enables or disables using connection pool.

Class

TCustomDAConnection

Syntax

```
property Pooling: boolean default DefValPooling;
```

Remarks

Normally, when TCustomDAConnection establishes connection with the server it takes server memory and time resources for allocating new server connection. For example, pooling can be very useful when using disconnect mode. If an application has wide user activity that forces many connect/disconnect operations, it may spend a lot of time on creating connection and sending requests to the server. TCustomDAConnection has software pool which stores open connections with identical parameters.

Connection pool uses separate thread that validates the pool every 30 seconds. Pool validation consists of checking each connection in the pool. If a connection is broken due to a network problem or another reason, it is deleted from the pool. The validation procedure removes also connections that are not used for a long time even if they are valid from the pool.

Set Pooling to True to enable pooling. Specify correct values for PoolingOptions. Two connections belong to the same pool if they have identical values for the parameters: MinPoolSize, MaxPoolSize, Validate, ConnectionLifeTime, TOraSession.Username, TOraSession.Server, TOraSession.ConnectMode, TOraSession.Options.

Note: Using Pooling := True can cause errors with working with temporary tables.

See Also

- Username
- Password
- PoolingOptions
- Using Connection Pooling

Request Support DAC Forum Provide Feedback

5.10.1.4.2.9 PoolingOptions Property

Specifies the behaviour of connection pool.

Class

TCustomDAConnection

Syntax

```
property PoolingOptions: TPoolingOptions;
```

Remarks

Set the properties of PoolingOptions to specify the behaviour of connection pool.

Descriptions of all options are in the table below.

Option Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- Pooling

5.10.1.4.2.10 Server Property

Serves to supply the server name for login.

Class

TCustomDAConnection

Syntax

```
property Server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login.

See Also

- Username
- Password

 Request Support DAC Forum Provide Feedback

5.10.1.4.2.11 Username Property

Used to supply a user name for login.

Class

TCustomDAConnection

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply a user name to handle server's request for login. If this property is not set, and OS authentication is used, a connection can be established. Otherwise an error is arised.

Warning: Storing hard-coded user name and password entries as property values or in code for the OnLogin event handler can compromise server security.

See Also

- Password
- Server

 Request Support DAC Forum Provide Feedback

5.10.1.4.3 Methods

Methods of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the

TCustomDAConnection Members topic.

Public

Name	Description
ApplyUpdates	Overloaded. Applies changes in datasets.
Commit	Commits current transaction.
Connect	Establishes a connection to the server.
CreateSQL	Creates a component for queries execution.
Disconnect	Performs disconnect.
ExecProc	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx	Allows to execute a stored procedure or function.
ExecSQL	Executes a SQL statement with parameters.
ExecSQLEx	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames	Returns a database list from the server.
GetKeyFieldNames	Provides a list of available key field names.
GetStoredProcNames	Returns a list of stored procedures from the server.
GetTableNames	Provides a list of available tables names.
MonitorMessage	Sends a specified message through the TCustomDASQLMonitor component.
Ping	Used to check state of connection to the server.
RemoveFromPool	Marks the connection that should not be returned to the pool after disconnect.

Rollback	Discards all current data changes and ends transaction.
StartTransaction	Begins a new user transaction.

See Also

- TCustomDAConnection Class
- TCustomDAConnection Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.4.3.1 ApplyUpdates Method

Applies changes in datasets.

Class

TCustomDAConnection

Overload List

Name	Description
ApplyUpdates	Applies changes from all active datasets.
ApplyUpdates(**const** DataSets: **array of** TCustomDADataSet)	Applies changes from the specified datasets.

 Request Support DAC Forum Provide Feedback

Applies changes from all active datasets.

Class

TCustomDAConnection

Syntax

```
procedure ApplyUpdates; overload; virtual;
```

Remarks

Call the ApplyUpdates method to write all pending cached updates from all active datasets attached to this connection to a database or from specific datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions, and clearing the cache when the operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than calling each individual dataset's ApplyUpdates method.

See Also

- TMemDataSet.CachedUpdates
- TMemDataSet.ApplyUpdates

Applies changes from the specified datasets.

Class

TCustomDAConnection

Syntax

```
procedure ApplyUpdates(const DataSets: array of TCustomDADataSet); overload; virtual;
```

Parameters

DataSets

A list of datasets changes in which are to be applied.

Remarks

Call the ApplyUpdates method to write all pending cached updates from the specified datasets. The ApplyUpdates method passes cached data to the database for storage, takes care of committing or rolling back transactions and clearing the cache when operation is successful.

Using ApplyUpdates for connection is a preferred method of updating datasets rather than

calling each individual dataset's ApplyUpdates method.

 Request Support DAC Forum Provide Feedback

5.10.1.4.3.2 Commit Method

Commits current transaction.

Class

TCustomDAConnection

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database and then ends the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- Rollback
- StartTransaction
- TOraDataSet.FetchAll

 Request Support DAC Forum Provide Feedback

5.10.1.4.3.3 Connect Method

Establishes a connection to the server.

Class

TCustomDAConnection

Syntax

```
procedure Connect; overload;procedure Connect(const
ConnectString: string); overload;
```

Remarks

Call the Connect method to establish a connection to the server. Connect sets the Connected property to True. If ConnectPrompt is True, Connect prompts user for login information as required by the server, or otherwise tries to establish a connection using values provided in the Username, Password, and Server properties.

If the Username and Password properties are not specified, then Oracle uses authentication information supplied at the operating system login process. For this feature to work make sure that your Oracle instance is appropriately tuned (see the Oracle documentation).

See Also

- Disconnect
- Username
- Password
- Server
- ConnectDialog

Request Support DAC Forum Provide Feedback

5.10.1.4.3.4 CreateSQL Method

Creates a component for queries execution.

Class

TCustomDAConnection

Syntax

```
function CreateSQL: TCustomDASQL; virtual;
```

Return Value

A new instance of the class.

Remarks

Call the CreateSQL to return a new instance of the TCustomDASQL class and associates it with this connection object. In the descendant classes this method should be overridden to create an appropriate descendant of the TCustomDASQL component.

5.10.1.4.3.5 Disconnect Method

Performs disconnect.

Class

TCustomDAConnection

Syntax

```
procedure Disconnect;
```

Remarks

Call the Disconnect method to drop a connection to database. Before the connection component is deactivated, all associated datasets are closed. Calling Disconnect is similar to setting the Connected property to False.

In most cases, closing a connection frees system resources allocated to the connection.

If user transaction is active, e.g. the InTransaction flag is set, calling to Disconnect commits the current user transaction.

Note: If a previously active connection is closed and then reopened, any associated datasets must be individually reopened; reopening the connection does not automatically reopen associated datasets.

See Also

- Connect

 Request Support DAC Forum Provide Feedback

5.10.1.4.3.6 ExecProc Method

Allows to execute stored procedure or function providing its name and parameters.

Class

TCustomDAConnection

Syntax

```
function ExecProc(const Name: string; const Params: array of variant): variant; virtual;
```

Parameters

Name

Holds the name of the stored procedure or function.

Params

Holds the parameters of the stored procedure or function.

Return Value

the result of the stored procedure.

Remarks

Allows to execute stored procedure or function providing its name and parameters.

Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign parameters' values to the Params array in exactly the same order and number as they appear in the stored procedure declaration. Out parameters of the procedure can be accessed with the ParamByName procedure.

If the value of an input parameter was not included to the Params array, parameter default value is taken. Only parameters at the end of the list can be unincluded to the Params array. If the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. The stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For further examples of parameter usage see ExecSQL, ExecSQLEx.

Example

For example, having stored function declaration presented in Example 1), you may execute it and retrieve its result with commands presented in Example 2):

```
Example 1)
CREATE procedure MY_SUM (
      A INTEGER,
      B INTEGER)
RETURNS (
      RESULT INTEGER)
as
begin
   Result = a + b;
end;
Example 2)
Label1.Caption:= MyOraConnection1.ExecProc('My_Sum', [10, 20]);
Label2.Caption:= MyOraConnection1.ParamByName('Result').AsString;
```

See Also

- ExecProcEx
- ExecSQL
- ExecSQLEx
- TOraSession.ParamByName
- TOraSession.SQL

5.10.1.4.3.7 ExecProcEx Method

Allows to execute a stored procedure or function.

Class

TCustomDAConnection

Syntax

```
function ExecProcEx(const Name: string; const Params: array of
variant): variant; virtual;
```

Parameters

Name

Holds the stored procedure name.

Params

Holds an array of pairs of parameters' names and values.

Return Value

the result of the stored procedure.

Remarks

Allows to execute a stored procedure or function. Provide the stored procedure name and its parameters to the call of ExecProcEx.

Use the following Name value syntax for executing specific overloaded routine: "StoredProcName:1" or "StoredProcName:5". The first example executes the first overloaded stored procedure, while the second example executes the fifth overloaded procedure.

Assign pairs of parameters' names and values to a Params array so that every name comes before its corresponding value when an array is being indexed.

Out parameters of the procedure can be accessed with the ParamByName procedure. If the value for an input parameter was not included to the Params array, the parameter default value is taken. If the parameter has no default value, the NULL value is sent.

Note: Stored functions unlike stored procedures return result values that are obtained internally through the RESULT parameter. You will no longer have to provide anonymous value in the Params array to describe the result of the function. Stored function result is obtained from the Params[0] indexed property or with the ParamByName('RESULT') method call.

For an example of parameters usage see ExecSQLEx.

Example

If you have some stored procedure accepting four parameters, and you want to provide values only for the first and fourth parameters, you should call ExecProcEx in the following way:

```
Connection.ExecProcEx('Some_Stored_Procedure', ['Param_Name1', 'Param_Value1
```

See Also

- ExecSQL
- ExecSQLEx
- ExecProc

5.10.1.4.3.8 ExecSQL Method

Executes a SQL statement with parameters.

Class

TCustomDAConnection

Syntax

```
function ExecSQL(const Text: string): variant;
overload;function ExecSQL(const Text: string; const Params:
array of variant): variant; overload; virtual;
```

Parameters

Text

a SQL statement to be executed.

Params

Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

Out parameter with the name Result will hold the result of function having data type dtString. Otherwise returns Null.

Remarks

Use the ExecSQL method to execute any SQL statement outside the TCustomDADataSet or TCustomDASQL components. Supply the Params array with the values of parameters arranged in the same order as they appear in a SQL statement which itself is passed to the Text string parameter.

The Params array must contain all IN and OUT parameters defined in SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to an ExecSQL method.

Out parameter with the name Result will hold the result of function having data type dtString. If none of the parameters in the Text statement is named Result, ExecSQL will return Null.

To get the values of OUT parameters use the ParamByName function.

Example

```
OraSession.ExecSQL('begin :A:= :B + :C; end;', [0, 5, 3]);
A:= OraSession.ParamByName('A').AsInteger;
```

See Also

- ExecSQLEx
- ExecProc
- TOraSession.SQL

5.10.1.4.3.9 ExecSQLEx Method

Executes any SQL statement outside the TQuery or TSQL components.

Class

TCustomDAConnection

Syntax

```
function ExecSQLEx(const Text: string; const Params: array of
variant): variant; virtual;
```

Parameters

Text

a SQL statement to be executed.

Params

Array of parameter values arranged in the same order as they appear in SQL statement.

Return Value

Out parameter with the name Result will hold the result of a function having data type dtString. Otherwise returns Null.

Remarks

Call the ExecSQLEx method to execute any SQL statement outside the TQuery or TSQL components. Supply the Params array with values arranged in pairs of parameter name and its value. This way each parameter name in the array is found on even index values whereas parameter value is on odd index value but right after its parameter name. The parameter pairs must be arranged according to their occurrence in a SQL statement which itself is passed in the Text string parameter.

The Params array must contain all IN and OUT parameters defined in the SQL statement. For OUT parameters provide any values of valid types so that they are explicitly defined before call to the ExecSQLEx method.

Out parameter with the name Result will hold the result of a function having data type dtString. If neither of the parameters in the Text statement is named Result, ExecSQLEx will return Null.

To get the values of OUT parameters use the ParamByName function.

Example

```
OraConnection.ExecSQLEx('begin :A:= :B + :C; end;',
   ['A', 0, 'B', 5, 'C', 3]);
A:= OraConnection.ParamByName('A').AsInteger;
```

See Also

- ExecSQL

Request Support DAC Forum Provide Feedback

5.10.1.4.3.10 GetDatabaseNames Method

Returns a database list from the server.

Class

TCustomDAConnection

Syntax

```
procedure GetDatabaseNames(List: TStrings); virtual;
```

Parameters

List

A TStrings descendant that will be filled with database names.

Remarks

Populates a string list with the names of databases.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetDatabaseNames.

See Also

- GetTableNames
- GetStoredProcNames

Request Support DAC Forum Provide Feedback

5.10.1.4.3.11 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDAConnection

Syntax

```
procedure GetKeyFieldNames(const TableName: string; List:
TStrings); virtual;
```

Parameters

TableName

Holds the table name

List

The list of available key field names

Return Value

Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

- GetTableNames
- GetStoredProcNames

 Request Support DAC Forum Provide Feedback

5.10.1.4.3.12 GetStoredProcNames Method

Returns a list of stored procedures from the server.

Class

TCustomDAConnection

Syntax

```
procedure GetStoredProcNames(List: TStrings; AllProcs: boolean = False); virtual;
```

Parameters

List

A TStrings descendant that will be filled with the names of stored procedures in the database.

AllProcs

True, if stored procedures from all schemas or including system procudures (depending on the server) are returned. False otherwise.

Remarks

Call the GetStoredProcNames method to get the names of available stored procedures and functions. GetStoredProcNames populates a string list with the names of stored procs in the database. If AllProcs = True, the procedure returns to the List parameter the names of the stored procedures that belong to all schemas; otherwise, List will contain the names of functions that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by data produced by GetStoredProcNames.

See Also

- GetDatabaseNames
- GetTableNames

5.10.1.4.3.13 GetTableNames Method

Provides a list of available tables names.

Class

TCustomDAConnection

Syntax

```
procedure GetTableNames(List: TStrings; AllTables: boolean = False; OnlyTables: boolean = False); virtual;
```

Parameters

List

A TStrings descendant that will be filled with table names.

AllTables

True, if procedure returns all table names including the names of system tables to the List parameter.

OnlyTables

Remarks

Call the GetTableNames method to get the names of available tables. Populates a string list with the names of tables in the database. If AllTables = True, procedure returns all table names including the names of system tables to the List parameter, otherwise List will not contain the names of system tables. If AllTables = True, the procedure returns to the List parameter the names of the tables that belong to all schemas; otherwise, List will contain the names of the tables that belong to the current schema.

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.

See Also

- GetDatabaseNames

- GetStoredProcNames

5.10.1.4.3.14 MonitorMessage Method

Sends a specified message through the TCustomDASQLMonitor component.

Class

TCustomDAConnection

Syntax

```
procedure MonitorMessage(const Msg: string);
```

Parameters

Msg

Message text that will be sent.

Remarks

Call the MonitorMessage method to output specified message via the TCustomDASQLMonitor component.

See Also

- TCustomDASQLMonitor

5.10.1.4.3.15 Ping Method

Used to check state of connection to the server.

Class

TCustomDAConnection

Syntax

```
procedure Ping;
```

Remarks

The method is used for checking server connection state.

Request Support DAC Forum Provide Feedback

5.10.1.4.3.16 RemoveFromPool Method

Marks the connection that should not be returned to the pool after disconnect.

Class

TCustomDAConnection

Syntax

```
procedure RemoveFromPool;
```

Remarks

Call the RemoveFromPool method to mark the connection that should be deleted after disconnect instead of returning to the connection pool.

See Also

- Pooling
- PoolingOptions

Request Support DAC Forum Provide Feedback

5.10.1.4.3.17 Rollback Method

Discards all current data changes and ends transaction.

Class

TCustomDAConnection

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call the Rollback method to discard all updates, insertions, and deletions of data associated with the current transaction to the database server and then end the transaction. The current transaction is the last transaction started by calling StartTransaction.

See Also

- Commit
- StartTransaction
- TOraDataSet.FetchAll

Request Support DAC Forum Provide Feedback

5.10.1.4.3.18 StartTransaction Method

Begins a new user transaction.

Class

TCustomDAConnection

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit or Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes, or Rollback to cancel them.

See Also

- Commit
- Rollback
- InTransaction

 Request Support DAC Forum Provide Feedback

5.10.1.4.4 Events

Events of the **TCustomDAConnection** class.

For a complete list of the **TCustomDAConnection** class members, see the TCustomDAConnection Members topic.

Public

Name	Description
OnConnectionLost	This event occurs when connection was lost.
OnError	This event occurs when an error has arisen in the connection.

See Also

- TCustomDAConnection Class
- TCustomDAConnection Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.4.4.1 OnConnectionLost Event

This event occurs when connection was lost.

Class

TCustomDAConnection

Syntax

```
property OnConnectionLost: TConnectionLostEvent;
```

Remarks

Write the OnConnectionLost event handler to process fatal errors and perform failover.

Note: To use the OnConnectionLost event handler, you should explicitly add the MemData unit to the 'uses' list and set the TCustomDAConnection.Options.LocalFailover property to True.

 Request Support DAC Forum Provide Feedback

5.10.1.4.4.2 OnError Event

This event occurs when an error has arisen in the connection.

Class

TCustomDAConnection

Syntax

```
property OnError: TDAConnectionErrorEvent;
```

Remarks

Write the OnError event handler to respond to errors that arise with connection. Check the E parameter to get the error code. Set the Fail parameter to False to prevent an error dialog from being displayed and to raise the EAbort exception to cancel current operation. The default value of Fail is True.

 Request Support DAC Forum Provide Feedback

5.10.1.5 TCustomDADataSet Class

Encapsulates general set of properties, events, and methods for working with data accessed through various database engines.

For a list of all members of this type, see TCustomDADataSet members.

Unit

DBAccess

Syntax

```
TCustomDADataSet = class(TMemDataSet);
```

Remarks

TCustomDADataSet encapsulates general set of properties, events, and methods for working with data accessed through various database engines. All database-specific features are supported by descendants of TCustomDADataSet.

Applications should not use TCustomDADataSet objects directly.

Inheritance Hierarchy

TMemDataSet

TCustomDADataSet

Request Support DAC Forum Provide Feedback

5.10.1.5.1 Members

TCustomDADataSet class overview.

Properties

Name	Description
BaseSQL	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Conditions	Used to add WHERE conditions to a query
Connection	Used to specify a connection object to use to connect to a data store.
DataTypeMap	Used to set data type mapping rules
Debug	Used to display the statement that is being

	executed and the values and types of its parameters.
DetailFields	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected	Used to keep dataset opened after connection is closed.
FetchRows	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.

MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
MasterFields	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource	Used to specify the data source component which binds current dataset to the master one.
Options	Used to specify the behaviour of TCustomDADataSet object.
ParamCheck	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount	Used to indicate how many parameters are there in the Params property.
Params	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

SQL	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLUpdate	Used to specify a SQL statement that will be used when applying an update to a dataset.
UniDirectional	Used if an application does not need bidirectional access to records in the result set.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description

AddWhere	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to find out whether

	TCustomDADataSet has fetched all rows.
Fetching	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey	Searches for a record which contains specified field values.
FindMacro	Finds a macro with the specified name.
FindNearest	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines if a parameter with the specified name exists in a dataset.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject	Returns a multireference shared object from field.
GetFieldPrecision	Retrieves the precision of a number field.
GetFieldScale	Retrieves the scale of a number field.
GetKeyFieldNames	Provides a list of available key field names.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.
GotoCurrent	Sets the current record in this dataset similar to the current record in another

	dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.

SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute	Occurs after a component has executed a query to database.
AfterFetch	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute	Occurs before executing

	insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.10.1.5.2 Properties

Properties of the **TCustomDADataSet** class.

For a complete list of the **TCustomDADataSet** class members, see the TCustomDADataSet Members topic.

Public

Name	Description
BaseSQL	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Conditions	Used to add WHERE conditions to a query
Connection	Used to specify a connection object to use to connect to a data store.
DataTypeMap	Used to set data type mapping rules
Debug	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields	Used to specify the fields that correspond to the

	foreign key fields from MasterFields when building master/detail relationship.
Disconnected	Used to keep dataset opened after connection is closed.
FetchRows	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsQuery	Used to check whether SQL statement returns rows.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MacroCount	Used to get the number of macros associated with the Macros property.

Macros	Makes it possible to change SQL queries easily.
MasterFields	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource	Used to specify the data source component which binds current dataset to the master one.
Options	Used to specify the behaviour of TCustomDADataSet object.
ParamCheck	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount	Used to indicate how many parameters are there in the Params property.
Params	Used to view and set parameter names, values, and data types dynamically.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshOptions	Used to indicate when the editing record is refreshed.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a query component executes when

	its Open method is called.
SQLDelete	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLUpdate	Used to specify a SQL statement that will be used when applying an update to a dataset.
UniDirectional	Used if an application does not need bidirectional access to records in the result set.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

5.10.1.5.2.1 BaseSQL Property

Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.

Class

TCustomDADataSet

Syntax

```
property BaseSQL: string;
```

Remarks

Use the BaseSQL property to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL, only macros are expanded. SQL text with all these changes can be returned by FinalSQL.

See Also

- FinalSQL
- AddWhere
- SaveSQL
- SQLSaved
- RestoreSQL

5.10.1.5.2.2 Conditions Property

Used to add WHERE conditions to a query

Class

TCustomDADataSet

Syntax

```
property Conditions: TDAConditions stored False;
```

See Also

- TDAConditions

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.3 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TCustomDADataSet

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.4 DataTypeMap Property

Used to set data type mapping rules

Class

TCustomDADataSet

Syntax

```
property DataTypeMap: TDAMapRules stored IsMapRulesStored;
```

See Also

- TDAMapRules

5.10.1.5.2.5 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TCustomDADataSet

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDASQL.Debug

5.10.1.5.2.6 DetailFields Property

Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.

Class

TCustomDADataSet

Syntax

```
property DetailFields: string;
```

Remarks

Use the DetailFields property to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship. DetailFields is a string containing one or more field names in the detail table. Separate field names with semicolons.

Use Field Link Designer to set the value in design time.

See Also

- MasterFields
- MasterSource

Request Support DAC Forum Provide Feedback

5.10.1.5.2.7 Disconnected Property

Used to keep dataset opened after connection is closed.

Class

TCustomDADataSet

Syntax

```
property Disconnected: boolean;
```

Remarks

Set the Disconnected property to True to keep dataset opened after connection is closed.

Request Support DAC Forum Provide Feedback

5.10.1.5.2.8 FetchRows Property

Used to define the number of rows to be transferred across the network at the same time.

Class

TCustomDADataSet

Syntax

```
property FetchRows: integer default 25;
```

Remarks

The number of rows that will be transferred across the network at the same time. This property can have a great impact on performance. So it is preferable to choose the optimal value of the FetchRows property for each SQL statement and software/hardware configuration experimentally.

The default value is 25.

5.10.1.5.2.9 FilterSQL Property

Used to change the WHERE clause of SELECT statement and reopen a query.

Class

TCustomDADataSet

Syntax

```
property FilterSQL: string;
```

Remarks

The FilterSQL property is similar to the Filter property, but it changes the WHERE clause of SELECT statement and reopens query. Syntax is the same to the WHERE clause.

Note: the FilterSQL property adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

Example

```
Query1.FilterSQL := 'Dept >= 20 and DName LIKE ''M%''';
```

See Also

- AddWhere

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.10 FinalSQL Property

Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.

Class

TCustomDADataSet

Syntax

```
property FinalSQL: string;
```

Remarks

Use FinalSQL to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros. This is the exact statement that will be passed on to the database server.

See Also

- FinalSQL
- AddWhere
- SaveSQL
- SQLSaved
- RestoreSQL
- BaseSQL

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.11 IsQuery Property

Used to check whether SQL statement returns rows.

Class

TCustomDADataSet

Syntax

```
property IsQuery: boolean;
```

Remarks

After the TCustomDADataSet component is prepared, the IsQuery property returns True if SQL statement is a SELECT query.

Use the IsQuery property to check whether the SQL statement returns rows or not.

IsQuery is a read-only property. Reading IsQuery on unprepared dataset raises an exception.

5.10.1.5.2.12 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database.

Class

TCustomDADataSet

Syntax

```
property KeyFields: string;
```

Remarks

TCustomDADataset uses the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating the database. For this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields is not defined before opening a dataset, TCustomDADataset requests information about primary keys from server sending an additional query.

Assign the KeyFields property with a string containing the name of a field which will be later assigned with Oracle sequenced values. Beforehand Oracle sequence must be created and its name passed to the TOraDataSet.KeySequence property.

Sequences are generated when either Insert or Post method is called. Which of these two methods is used to modify the database is determined by the TOraDataSet.SequenceMode property.

Note: Though keys may be created across a number of table fields, sequence is generated only for the first field found in the KeyFields property.

See Also

- SQLDelete
- SQLInsert
- SQLRefresh
- SQLUpdate
- TOraDataSet.KeySequence
- TOraDataSet.SequenceMode

Request Support DAC Forum Provide Feedback

5.10.1.5.2.13 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

TCustomDADataSet

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

Request Support DAC Forum Provide Feedback

5.10.1.5.2.14 Macros Property

Makes it possible to change SQL queries easily.

Class

TCustomDADataSet

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

Example

```
OraQuery.SQL.Text := 'SELECT * FROM Dept ORDER BY &Order';
OraQuery.MacroByName('Order').Value:= 'DeptNo';
OraQuery.Open;
```

See Also

- TMacro
- MacroByName
- Params

Request Support DAC Forum Provide Feedback

5.10.1.5.2.15 MasterFields Property

Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

Class

TCustomDADataSet

Syntax

```
property MasterFields: string;
```

Remarks

Use the MasterFields property after setting the MasterSource property to specify the names of one or more fields that are used as foreign keys for this dataset when establishing detail/ master relationship between it and the dataset specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in these fields are used to select corresponding records in this table for display.

Use Field Link Designer to set the values at design time after setting the MasterSource property.

See Also

- DetailFields
- MasterSource
- Master/Detail Relationships

5.10.1.5.2.16 MasterSource Property

Used to specify the data source component which binds current dataset to the master one.

Class

TCustomDADataSet

Syntax

```
property MasterSource: TDataSource;
```

Remarks

The MasterSource property specifies the data source component which binds current dataset to the master one.

TCustomDADataset uses MasterSource to extract foreign key fields values from the master dataset when building master/detail relationship between two datasets. MasterSource must point to another dataset; it cannot point to this dataset component.

When MasterSource is not **nil** dataset fills parameter values with corresponding field values from the current record of the master dataset.

Note: Do not set the DataSource property when building master/detail relationships. Although it points to the same object as the MasterSource property, it may lead to undesirable results.

See Also

- MasterFields
- DetailFields
- Master/Detail Relationships

5.10.1.5.2.17 Options Property

Used to specify the behaviour of TCustomDADataSet object.

Class

TCustomDADataSet

Syntax

```
property Options: TDADataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TCustomDADataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoPrepare	Used to execute automatic Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue

	properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also

- Master/Detail Relationships
- TMemDataSet.CachedUpdates

5.10.1.5.2.18 ParamCheck Property

Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Class

TCustomDADataSet

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

Set ParamCheck to True to let dataset automatically generate the Params property for the dataset based on a SQL statement.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of stored procedures which themselves will accept parameterized values. The default value is True.

See Also

- Params

Request Support DAC Forum Provide Feedback

5.10.1.5.2.19 ParamCount Property

Used to indicate how many parameters are there in the Params property.

Class

TCustomDADataSet

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

See Also

- Params

Request Support DAC Forum Provide Feedback

5.10.1.5.2.20 Params Property

Used to view and set parameter names, values, and data types dynamically.

Class

TCustomDADataSet

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set the parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also

- ParamByName
- Macros

Request Support DAC Forum Provide Feedback

5.10.1.5.2.21 ReadOnly Property

Used to prevent users from updating, inserting, or deleting data in the dataset.

Class

TCustomDADataSet

Syntax

```
property ReadOnly: boolean default False;
```

Remarks

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is False, meaning that users can potentially alter data stored in the dataset.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to True.

When ReadOnly is True, the dataset's CanModify property is False.

Request Support DAC Forum Provide Feedback

5.10.1.5.2.22 RefreshOptions Property

Used to indicate when the editing record is refreshed.

Class

TCustomDADataSet

Syntax

```
property RefreshOptions: TRefreshOptions default [];
```

Remarks

Use the RefreshOptions property to determine when the editing record is refreshed.

Refresh is performed by the RefreshRecord method.

It queries the current record and replaces one in the dataset. Refresh record is useful when the table has triggers or the table fields have default values. Use roBeforeEdit to get actual

data before editing.

The default value is [].

See Also

- RefreshRecord

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.23 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDADataSet

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.24 SQL Property

Used to provide a SQL statement that a query component executes when its Open method is called.

Class

TCustomDADataSet

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a query component executes when its Open method is called. At the design time the SQL property can be edited by invoking the String List editor in Object Inspector.

When SQL is changed, TCustomDADataSet calls Close and UnPrepare.

See Also

- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

Request Support DAC Forum Provide Feedback

5.10.1.5.2.25 SQLDelete Property

Used to specify a SQL statement that will be used when applying a deletion to a record.

Class

TCustomDADataSet

Syntax

```
property SQLDelete: TStrings;
```

Remarks

Use the SQLDelete property to specify the SQL statement that will be used when applying a deletion to a record. Statements can be parameterized queries.

To create a SQLDelete statement at design-time, use the query statements editor.

Example

```
DELETE FROM Orders
   WHERE
```

```
      OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
- SQLUpdate
- SQLRefresh

 Request Support DAC Forum Provide Feedback

5.10.1.5.2.26 SQLInsert Property

Used to specify the SQL statement that will be used when applying an insertion to a dataset.

Class

TCustomDADataSet

Syntax

```
property SQLInsert: TStrings;
```

Remarks

Use the SQLInsert property to specify the SQL statement that will be used when applying an insertion to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. Parameters prefixed with OLD_ allow using current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to dataset.

To create a SQLInsert statement at design-time, use the query statements editor.

See Also

- SQL
- SQLUpdate
- SQLDelete
- SQLRefresh

5.10.1.5.2.27 SQLLock Property

Used to specify a SQL statement that will be used to perform a record lock.

Class

TCustomDADataSet

Syntax

```
property SQLLock: TStrings;
```

Remarks

Use the SQLLock property to specify a SQL statement that will be used to perform a record lock. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

To create a SQLLock statement at design-time, the use query statement editor.

See Also

- SQL
- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh

5.10.1.5.2.28 SQLRecCount Property

Used to specify the SQL statement that is used to get the record count when opening a dataset.

Class

TCustomDADataSet

Syntax

```
property SQLRecCount: TStrings;
```

Remarks

Use the SQLRecCount property to specify the SQL statement that is used to get the record count when opening a dataset. The SQL statement is used if the TDADataSetOptions.QueryRecCount property is True, and the TCustomDADataSet.FetchAll property is False. Is not used if the FetchAll property is True.

To create a SQLRecCount statement at design-time, use the query statements editor.

See Also

- SQLInsert
- SQLUpdate
- SQLDelete
- SQLRefresh
- TDADataSetOptions
- FetchingAll

Request Support DAC Forum Provide Feedback

5.10.1.5.2.29 SQLRefresh Property

Used to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Class

TCustomDADataSet

Syntax

```
property SQLRefresh: TStrings;
```

Remarks

Use the SQLRefresh property to specify a SQL statement that will be used to refresh current record by calling the RefreshRecord procedure.

Different behavior is observed when the SQLRefresh property is assigned with a single WHERE clause that holds frequently altered search condition. In this case the WHERE clause from SQLRefresh is combined with the same clause of the SELECT statement in a SQL property and this final query is then sent to the database server.

To create a SQLRefresh statement at design-time, use the query statements editor.

Example

```
SELECT Shipname FROM Orders
   WHERE
      OrderID = :OrderID
```

See Also

- RefreshRecord
- SQL
- SQLInsert
- SQLUpdate
- SQLDelete

5.10.1.5.2.30 SQLUpdate Property

Used to specify a SQL statement that will be used when applying an update to a dataset.

Class

TCustomDADataSet

Syntax

```
property SQLUpdate: TStrings;
```

Remarks

Use the SQLUpdate property to specify a SQL statement that will be used when applying an update to a dataset. Statements can be parameterized queries. Names of the parameters should be the same as field names. The parameters prefixed with OLD_ allow to use current values of fields prior to the actual operation.

Use ReturnParam to return OUT parameters back to the dataset.

To create a SQLUpdate statement at design-time, use the query statement editor.

Example

```
UPDATE Orders
   set
      ShipName = :ShipName
   WHERE
      OrderID = :Old_OrderID
```

See Also

- SQL
- SQLInsert
- SQLDelete
- SQLRefresh

5.10.1.5.2.31 UniDirectional Property

Used if an application does not need bidirectional access to records in the result set.

Class

TCustomDADataSet

Syntax

```
property UniDirectional: boolean default False;
```

Remarks

Traditionally SQL cursors are unidirectional. They can travel only forward through a dataset. TCustomDADataset, however, permits bidirectional travelling by caching records. If an

application does not need bidirectional access to the records in the result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved. However, UniDirectional datasets cannot be modified. In FetchAll=False mode data is fetched on demand. When UniDirectional is set to True, data is fetched on demand as well, but obtained rows are not cached except for the current row. In case if the Unidirectional property is True, the FetchAll property will be automatically set to False. And if the FetchAll property is True, the Unidirectional property will be automatically set to False. The default value of UniDirectional is False, enabling forward and backward navigation.

Note: Pay attention to the specificity of using the FetchAll property=False

See Also

- TOraDataSet.FetchAll

 Request Support DAC Forum Provide Feedback

5.10.1.5.3 Methods

Methods of the **TCustomDADataSet** class.

For a complete list of the **TCustomDADataSet** class members, see the TCustomDADataSet Members topic.

Public

Name	Description
AddWhere	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a

	dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Indicates whether SQL statement is still being executed.
Fetched	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey	Searches for a record which contains specified field values.
FindMacro	Finds a macro with the specified name.

FindNearest	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines if a parameter with the specified name exists in a dataset.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType	Returns internal field types defined in the MemData and accompanying modules.
GetFieldObject	Returns a multireference shared object from field.
GetFieldPrecision	Retrieves the precision of a number field.
GetFieldScale	Retrieves the scale of a number field.
GetKeyFieldNames	Provides a list of available key field names.
GetOrderBy	Retrieves an ORDER BY clause from a SQL statement.
GotoCurrent	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock	Locks the current record.
MacroByName	Finds a macro with the specified name.

ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare	Allocates, opens, and parses cursor for a query.
RefreshRecord	Actualizes field values for the current record.
RestoreSQL	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved	Determines if the SQL property value was saved to the BaseSQL property.
UnLock	Releases a record lock.

UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

Request Support DAC Forum Provide Feedback

5.10.1.5.3.1 AddWhere Method

Adds condition to the WHERE clause of SELECT statement in the SQL property.

Class

TCustomDADataSet

Syntax

```
procedure AddWhere(const Condition: string);
```

Parameters

Condition

Holds the condition that will be added to the WHERE clause.

Remarks

Call the AddWhere method to add a condition to the WHERE clause of SELECT statement in the SQL property.

If SELECT has no WHERE clause, AddWhere creates it.

Note: the AddWhere method is implicitly called by RefreshRecord. The AddWhere method works for the SELECT statements only.

Note: the AddWhere method adds a value to the WHERE condition as is. If you expect this value to be enclosed in brackets, you should bracket it explicitly.

See Also

- DeleteWhere

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.2 BreakExec Method

Breaks execution of a SQL statement on the server.

Class

TCustomDADataSet

Syntax

```
procedure BreakExec; virtual;
```

Remarks

Call the BreakExec method to break execution of a SQL statement on the server. Useful when TOraDataSet.NonBlocking is True.

There are some notions to keep in mind when using this procedure:

- execution of the PL/SQL block cannot be interrupted by BreakExec;
- calling BreakExec to interrupt dataset opening in the TOraDataSet.NonBlocking mode may not have effect if a fetch operation has already begun (this happens when BreakExec falls between two fetch operations).

See Also

- TCustomDADataSet.Execute
- TOraDataSet.NonBlocking

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.3 CreateBlobStream Method

Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.

Class

TCustomDADataSet

Syntax

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode):
TStream; override;
```

Parameters

Field

Holds the BLOB field for reading data from or writing data to from a stream.

Mode

Holds the stream mode, for which the stream will be used.

Return Value

The BLOB Stream.

Remarks

Call the CreateBlobStream method to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter. It must be a TBlobField component. You can specify whether the stream will be used for reading, writing, or updating the contents of the field with the Mode parameter.

5.10.1.5.3.4 DeleteWhere Method

Removes WHERE clause from the SQL property and assigns the BaseSQL property.

Class

TCustomDADataSet

Syntax

```
procedure DeleteWhere;
```

Remarks

Call the DeleteWhere method to remove WHERE clause from the the SQL property and assign BaseSQL.

See Also

- AddWhere
- BaseSQL

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.5 Execute Method

Executes a SQL statement on the server.

Class

TCustomDADataSet

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

 Request Support DAC Forum Provide Feedback

Executes a SQL statement on the server.

Class

TCustomDADataSet

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

Call the Execute method to execute an SQL statement on the server. If SQL statement is a SELECT query, Execute calls the Open method.

Execute implicitly prepares SQL statement by calling the TCustomDADataSet.Prepare method if the TCustomDADataSet.Options option is set to True and the statement has not been prepared yet. To speed up the performance in case of multiple Execute calls, an application should call Prepare before calling the Execute method for the first time.

See Also

- TCustomDADataSet.AfterExecute
- TCustomDADataSet.Executing
- TCustomDADataSet.Prepare

 Request Support DAC Forum Provide Feedback

Used to perform Batch operations .

Class

TCustomDADataSet

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters

Specifies the number of inserted rows.

Offset

Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations

article for samples.

See Also

- Batch operations

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.6 Executing Method

Indicates whether SQL statement is still being executed.

Class

TCustomDADataSet

Syntax

```
function Executing: boolean;
```

Return Value

True, if SQL statement is still being executed.

Remarks

Check Executing to learn whether TCustomDADataSet is still executing SQL statement. Use the Executing method if NonBlocking is True.

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.7 Fetched Method

Used to find out whether TCustomDADataSet has fetched all rows.

Class

TCustomDADataSet

Syntax

```
function Fetched: boolean; virtual;
```

Return Value

True, if all rows have been fetched.

Remarks

Call the Fetched method to find out whether TCustomDADataSet has fetched all rows.

See Also

- Fetching

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.8 Fetching Method

Used to learn whether TCustomDADataSet is still fetching rows.

Class

TCustomDADataSet

Syntax

```
function Fetching: boolean;
```

Return Value

True, if TCustomDADataSet is still fetching rows.

Remarks

Check Fetching to learn whether TCustomDADataSet is still fetching rows. Use the Fetching method if NonBlocking is True.

See Also

- Executing

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.9 FetchingAll Method

Used to learn whether TCustomDADataSet is fetching all rows to the end.

Class

TCustomDADataSet

Syntax

```
function FetchingAll: boolean;
```

Return Value

True, if TCustomDADataSet is fetching all rows to the end.

Remarks

Check FetchingAll to learn whether TCustomDADataSet is fetching all rows to the end.

See Also

- Executing

Request Support DAC Forum Provide Feedback

5.10.1.5.3.10 FindKey Method

Searches for a record which contains specified field values.

Class

TCustomDADataSet

Syntax

```
function FindKey(const KeyValues: array of System.TVarRec):
Boolean;
```

Parameters

KeyValues

Holds a key.

Remarks

Call the FindKey method to search for a specific record in a dataset. KeyValues holds a comma-delimited array of field values, that is called a key.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

5.10.1.5.3.11 FindMacro Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

```
function FindMacro(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro
- Macros
- MacroByName

5.10.1.5.3.12 FindNearest Method

Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.

Class

TCustomDADataSet

Syntax

```
procedure FindNearest(const KeyValues: array of System.TVarRec);
```

Parameters

KeyValues

Holds the values of the record key fields to which the cursor should be moved.

Remarks

Call the FindNearest method to move the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter. If there are no records that match or exceed the specified criteria, the cursor will not move.

This function is provided for BDE compatibility only. It is recommended to use functions TMemDataSet.Locate and TMemDataSet.LocateEx for the record search.

See Also

- TMemDataSet.Locate
- TMemDataSet.LocateEx
- FindKey

5.10.1.5.3.13 FindParam Method

Determines if a parameter with the specified name exists in a dataset.

Class

TCustomDADataSet

Syntax

```
function FindParam(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the param for which to search.

Return Value

the TDAParam object for the specified Name. Otherwise it returns nil.

Remarks

Call the FindParam method to determine if a specified param component exists in a dataset. Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TDAParam object for the specified Name. Otherwise it returns nil.

See Also

- Params
- ParamByName

Request Support DAC Forum Provide Feedback

5.10.1.5.3.14 GetDataType Method

Returns internal field types defined in the MemData and accompanying modules.

Class

TCustomDADataSet

Syntax

```
function GetDataType(const FieldName: string): integer; virtual;
```

Parameters

FieldName

Holds the name of the field.

Return Value

internal field types defined in MemData and accompanying modules.

Remarks

Call the GetDataType method to return internal field types defined in the MemData and accompanying modules. Internal field data types extend the TFieldType type of VCL by specific database server data types. For example, ftString, ftFile, ftObject.

Request Support DAC Forum Provide Feedback

Reserved.

5.10.1.5.3.15 GetFieldObject Method

Returns a multireference shared object from field.

Class

TCustomDADataSet

Syntax

```
function GetFieldObject(Field: TField): TSharedObject;
overload;function GetFieldObject(Field: TField; RecBuf:
TRecordBuffer): TSharedObject; overload;function
GetFieldObject(FieldDesc: TFieldDesc): TSharedObject;
overload;function GetFieldObject(FieldDesc: TFieldDesc; RecBuf:
TRecordBuffer): TSharedObject; overload;function
GetFieldObject(const FieldName: string): TSharedObject; overload;
```

Parameters

FieldName

Holds the field name.

Return Value

multireference shared object.

Remarks

Call the GetFieldObject method to return a multireference shared object from field. If field does not hold one of the TSharedObject descendants, GetFieldObject raises an exception.

Request Support DAC Forum Provide Feedback

5.10.1.5.3.16 GetFieldPrecision Method

Retrieves the precision of a number field.

Class

TCustomDADataSet

Syntax

```
function GetFieldPrecision(const FieldName: string): integer;
```

Parameters

FieldName

Holds the existing field name.

Return Value

precision of number field.

Remarks

Call the GetFieldPrecision method to retrieve the precision of a number field. FieldName is the name of an existing field.

See Also

- GetFieldScale

Request Support DAC Forum Provide Feedback

5.10.1.5.3.17 GetFieldScale Method

Retrieves the scale of a number field.

Class

TCustomDADataSet

Syntax

```
function GetFieldScale(const FieldName: string): integer;
```

Parameters

FieldName

Holds the existing field name.

Return Value

the scale of the number field.

Remarks

Call the GetFieldScale method to retrieve the scale of a number field. FieldName is the name of an existing field.

See Also

- GetFieldPrecision

5.10.1.5.3.18 GetKeyFieldNames Method

Provides a list of available key field names.

Class

TCustomDADataSet

Syntax

```
procedure GetKeyFieldNames(List: TStrings);
```

Parameters

List

The list of available key field names

Return Value

Key field name

Remarks

Call the GetKeyFieldNames method to get the names of available key fields. Populates a string list with the names of key fields in tables.

See Also

- TCustomDAConnection.GetTableNames
- TCustomDAConnection.GetStoredProcNames

5.10.1.5.3.19 GetOrderBy Method

Retrieves an ORDER BY clause from a SQL statement.

Class

TCustomDADataSet

Syntax

```
function GetOrderBy: string;
```

Return Value

an ORDER BY clause from the SQL statement.

Remarks

Call the GetOrderBy method to retrieve an ORDER BY clause from a SQL statement.

Note: GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

See Also

- SetOrderBy

5.10.1.5.3.20 GotoCurrent Method

Sets the current record in this dataset similar to the current record in another dataset.

Class

TCustomDADataSet

Syntax

```
procedure GotoCurrent(DataSet: TCustomDADataSet);
```

Parameters

DataSet

Holds the TCustomDADataSet descendant to synchronize the record position with.

Remarks

Call the GotoCurrent method to set the current record in this dataset similar to the current record in another dataset. The key fields in both these DataSets must be coincident.

See Also

- TMemDataSet.Locate
- TMemDataSet.LocateEx

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.21 Lock Method

Locks the current record.

Class

TCustomDADataSet

Syntax

```
procedure Lock; virtual;
```

Remarks

Call the Lock method to lock the current record by executing the statement that is defined in the SQLLock property (for the TOraQuery component).

The Lock method sets the savepoint with the name LOCK_ + <component_name>.

TOraQuery uses SQLLock to execute the current record locking. TSmartQuery builds a SELECT FOR UPDATE statement itself.

See Also

- TOraDataSet.LockMode
- SQLLock
- UnLock

 Request Support DAC Forum Provide Feedback

5.10.1.5.3.22 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDADataSet

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

Example

```
OraQuery.SQL:= 'SELECT * FROM Scott.Dept ORDER BY &Order';
OraQuery.MacroByName('Order').Value:= 'DeptNo';
OraQuery.Open;
```

See Also

- TMacro
- Macros
- FindMacro

5.10.1.5.3.23 ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class

TCustomDADataSet

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the parameter for which to retrieve information.

Return Value

a TDAParam object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TDAParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- Params
- FindParam

5.10.1.5.3.24 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDADataSet

Syntax

```
procedure Prepare; override;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

TCustomDADataSet automatically prepares a query if it is executed without being prepared first. After execution, TCustomDADataSet unprepares the query. When a query is executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- TMemDataSet.Prepared
- TMemDataSet.UnPrepare
- Options

5.10.1.5.3.25 RefreshRecord Method

Actualizes field values for the current record.

Class

TCustomDADataSet

Syntax

```
procedure RefreshRecord;
```

Remarks

Call the RefreshRecord method to actualize field values for the current record. RefreshRecord performs query to database and refetches new field values from the returned cursor.

See Also

- RefreshOptions
- SQLRefresh

Request Support DAC Forum Provide Feedback

5.10.1.5.3.26 RestoreSQL Method

Restores the SQL property modified by AddWhere and SetOrderBy.

Class

TCustomDADataSet

Syntax

```
procedure RestoreSQL;
```

Remarks

Call the RestoreSQL method to restore the SQL property modified by AddWhere and SetOrderBy.

See Also

- AddWhere
- SetOrderBy
- SaveSQL
- SQLSaved

5.10.1.5.3.27 SaveSQL Method

Saves the SQL property value to BaseSQL.

Class

TCustomDADataSet

Syntax

```
procedure SaveSQL;
```

Remarks

Call the SaveSQL method to save the SQL property value to the BaseSQL property.

See Also

- SQLSaved
- RestoreSQL
- BaseSQL

5.10.1.5.3.28 SetOrderBy Method

Builds an ORDER BY clause of a SELECT statement.

Class

TCustomDADataSet

Syntax

```
procedure SetOrderBy(const Fields: string);
```

Parameters

Fields

Holds the names of the fields which will be added to the ORDER BY clause.

Remarks

Call the SetOrderBy method to build an ORDER BY clause of a SELECT statement. The fields are identified by the comma-delimited field names.

Note: The GetOrderBy and SetOrderBy methods serve to process only quite simple queries and don't support, for example, subqueries.

Example

```
Query1.SetOrderBy('DeptNo;DName');
```

See Also

- GetOrderBy

Request Support DAC Forum Provide Feedback

5.10.1.5.3.29 SQLSaved Method

Determines if the SQL property value was saved to the BaseSQL property.

Class

TCustomDADataSet

Syntax

```
function SQLSaved: boolean;
```

Return Value

True, if the SQL property value was saved to the BaseSQL property.

Remarks

Call the SQLSaved method to know whether the SQL property value was saved to the BaseSQL property.

Request Support DAC Forum Provide Feedback

5.10.1.5.3.30 UnLock Method

Releases a record lock.

Class

TCustomDADataSet

Syntax

```
procedure UnLock;
```

Remarks

Call the Unlock method to release the record lock made by the Lock method before.

Unlock is performed by rolling back to the savepoint set by the Lock method.

See Also

- Lock

Request Support DAC Forum Provide Feedback

5.10.1.5.4 Events

Events of the **TCustomDADataSet** class.

For a complete list of the **TCustomDADataSet** class members, see the TCustomDADataSet Members topic.

Public

Name	Description
AfterExecute	Occurs after a component has executed a query to database.
AfterFetch	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch	Occurs before dataset is going to fetch block of

	records from the server.
BeforeUpdateExecute	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

See Also

- TCustomDADataSet Class
- TCustomDADataSet Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.5.4.1 AfterExecute Event

Occurs after a component has executed a query to database.

Class

TCustomDADataSet

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a component has executed a query to database.

See Also

- TCustomDADataSet.Execute

 Request Support DAC Forum Provide Feedback

5.10.1.5.4.2 AfterFetch Event

Occurs after dataset finishes fetching data from server.

Class

TCustomDADataSet

Syntax

```
property AfterFetch: TAfterFetchEvent;
```

Remarks

The AfterFetch event occurs after dataset finishes fetching data from server.

See Also

- BeforeFetch
- TOraDataSet.NonBlocking

 Request Support DAC Forum Provide Feedback

5.10.1.5.4.3 AfterUpdateExecute Event

Occurs after executing insert, delete, update, lock and refresh operations.

Class

TCustomDADataSet

Syntax

```
property AfterUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs after executing insert, delete, update, lock, and refresh operations. You can use AfterUpdateExecute to set the parameters of corresponding statements.

 Request Support DAC Forum Provide Feedback

5.10.1.5.4.4 BeforeFetch Event

Occurs before dataset is going to fetch block of records from the server.

Class

TCustomDADataSet

Syntax

```
property BeforeFetch: TBeforeFetchEvent;
```

Remarks

The BeforeFetch event occurs every time before dataset is going to fetch a block of records from the server. Set Cancel to True to abort current fetch operation.

Note: In the TOraDataSet.NonBlocking mode event handler is called from the fetching thread. Therefore, if you have set the NonBlocking property to True, you should use thread synchronization mechanisms in the code of the BeforeFetch event handler.

See Also

- AfterFetch
- TOraDataSet.NonBlocking

5.10.1.5.4.5 BeforeUpdateExecute Event

Occurs before executing insert, delete, update, lock, and refresh operations.

Class

TCustomDADataSet

Syntax

```
property BeforeUpdateExecute: TUpdateExecuteEvent;
```

Remarks

Occurs before executing insert, delete, update, lock, and refresh operations. You can use

BeforeUpdateExecute to set the parameters of corresponding statements.

See Also

- AfterUpdateExecute

 Request Support DAC Forum Provide Feedback

5.10.1.6 TCustomDASQL Class

A base class for components executing SQL statements that do not return result sets.

For a list of all members of this type, see TCustomDASQL members.

Unit

DBAccess

Syntax

```
TCustomDASQL = class(TComponent);
```

Remarks

TCustomDASQL is a base class that defines functionality for descendant classes which access database using SQL statements. Applications never use TCustomDASQL objects directly. Instead they use descendants of TCustomDASQL.

Use TCustomDASQL when client application must execute SQL statement or call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

 Request Support DAC Forum Provide Feedback

5.10.1.6.1 Members

TCustomDASQL class overview.

Properties

Name	Description

ChangeCursor	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL	Used to return a SQL statement with expanded macros.
MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
ParamCheck	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount	Indicates the number of parameters in the Params property.
Params	Used to contain parameters for a SQL statement.
ParamValues	Used to get or set the values of individual field parameters that are identified by name.
Prepared	Used to indicate whether a query is prepared for execution.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a TCustomDASQL component executes when

	the Execute method is called.

Methods

Name	Description
BreakExec	Breaks execution of an SQL satatement on the server.
Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Finds a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a macro with the specified name.
ParamByName	Finds a parameter with the specified name.
Prepare	Allocates, opens, and parses cursor for a query.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

5.10.1.6.2 Properties

Properties of the **TCustomDASQL** class.

For a complete list of the **TCustomDASQL** class members, see the TCustomDASQL Members topic.

Public

Name	Description
ChangeCursor	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection	Used to specify a connection object to use to connect to a data store.
Debug	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL	Used to return a SQL statement with expanded macros.
MacroCount	Used to get the number of macros associated with the Macros property.
Macros	Makes it possible to change SQL queries easily.
ParamCheck	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount	Indicates the number of parameters in the Params property.
Params	Used to contain parameters for a SQL statement.
ParamValues	Used to get or set the values of individual field parameters that are identified by name.
Prepared	Used to indicate whether a

	query is prepared for execution.
RowsAffected	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
SQL	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

See Also

- TCustomDASQL Class
- TCustomDASQL Class Members

Request Support DAC Forum Provide Feedback

5.10.1.6.2.1 ChangeCursor Property

Enables or disables changing screen cursor when executing commands in the NonBlocking mode.

Class

TCustomDASQL

Syntax

```
property ChangeCursor: boolean;
```

Remarks

Set the ChangeCursor property to False to prevent the screen cursor from changing to crSQLArrow when executing commands in the NonBlocking mode. The default value is True.

Request Support DAC Forum Provide Feedback

5.10.1.6.2.2 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TCustomDASQL

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object that will be used to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, link an instance of a TCustomDAConnection descendant to the Connection property.

5.10.1.6.2.3 Debug Property

Used to display the statement that is being executed and the values and types of its parameters.

Class

TCustomDASQL

Syntax

```
property Debug: boolean default False;
```

Remarks

Set the Debug property to True to display the statement that is being executed and the values and types of its parameters.

You should add the OdacVcl unit to the uses clause of any unit in your project to make the Debug property work.

Note: If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

See Also

- TCustomDADataSet.Debug

 Request Support DAC Forum Provide Feedback

5.10.1.6.2.4 FinalSQL Property

Used to return a SQL statement with expanded macros.

Class

TCustomDASQL

Syntax

```
property FinalSQL: string;
```

Remarks

Read the FinalSQL property to return a SQL statement with expanded macros. This is the exact statement that will be passed on to the database server.

 Request Support DAC Forum Provide Feedback

5.10.1.6.2.5 MacroCount Property

Used to get the number of macros associated with the Macros property.

Class

TCustomDASQL

Syntax

```
property MacroCount: word;
```

Remarks

Use the MacroCount property to get the number of macros associated with the Macros property.

See Also

- Macros

Request Support DAC Forum Provide Feedback

5.10.1.6.2.6 Macros Property

Makes it possible to change SQL queries easily.

Class

TCustomDASQL

Syntax

```
property Macros: TMacros stored False;
```

Remarks

With the help of macros you can easily change SQL query text at design- or runtime. Marcos extend abilities of parameters and allow to change conditions in a WHERE clause or sort order in an ORDER BY clause. You just insert &MacroName in the SQL query text and change value of macro in the Macro property editor at design time or call the MacroByName function at run time. At the time of opening the query macro is replaced by its value.

See Also

- TMacro
- MacroByName
- Params

Request Support DAC Forum Provide Feedback

5.10.1.6.2.7 ParamCheck Property

Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Class

TCustomDASQL

Syntax

```
property ParamCheck: boolean default True;
```

Remarks

Use the ParamCheck property to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.

Set ParamCheck to True to let TCustomDASQL generate the Params property for the dataset based on a SQL statement automatically.

Setting ParamCheck to False can be used if the dataset component passes to a server the DDL statements that contain, for example, declarations of the stored procedures that will accept parameterized values themselves. The default value is True.

See Also

- Params

Request Support DAC Forum Provide Feedback

5.10.1.6.2.8 ParamCount Property

Indicates the number of parameters in the Params property.

Class

TCustomDASQL

Syntax

```
property ParamCount: word;
```

Remarks

Use the ParamCount property to determine how many parameters are there in the Params property.

Request Support DAC Forum Provide Feedback

5.10.1.6.2.9 Params Property

Used to contain parameters for a SQL statement.

Class

TCustomDASQL

Syntax

```
property Params: TDAParams stored False;
```

Remarks

Access the Params property at runtime to view and set parameter names, values, and data types dynamically (at design-time use the Parameters editor to set parameter properties). Params is a zero-based array of parameter records. Index specifies the array element to access. An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters at runtime:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with OraSQL do
 begin
      SQL.Clear;
      SQL.Add('INSERT INTO Temp_Table(Id, Name)');
      SQL.Add('VALUES (:id, :Name)');
      ParamByName('Id').AsInteger := 55;
      Params[1].AsString := ' Green';
      Execute;
   end;
end;
```

See Also

- TDAParam
- FindParam
- Macros

 Request Support DAC Forum Provide Feedback

5.10.1.6.2.10 ParamValues Property(Indexer)

Used to get or set the values of individual field parameters that are identified by name.

Class

TCustomDASQL

Syntax

```
property ParamValues[const ParamName: string]: Variant; default;
```

Parameters

ParamName

Holds parameter names separated by semicolon.

Remarks

Use the ParamValues property to get or set the values of individual field parameters that are identified by name.

Setting ParamValues sets the Value property for each parameter listed in the ParamName string. Specify the values as Variants.

Getting ParamValues retrieves an array of variants, each of which represents the value of one of the named parameters.

Note: The Params array is generated implicitly if ParamCheck property is set to True. If ParamName includes a name that does not match any of the parameters in Items, an exception is raised.

 Request Support DAC Forum Provide Feedback

5.10.1.6.2.11 Prepared Property

Used to indicate whether a query is prepared for execution.

Class

TCustomDASQL

Syntax

```
property Prepared: boolean;
```

Remarks

Check the Prepared property to determine if a query is already prepared for execution. True means that the query has already been prepared. As a rule prepared queries are executed faster, but the preparation itself also takes some time. One of the proper cases for using preparation is parametrized queries that are executed several times.

See Also

- Prepare

Request Support DAC Forum Provide Feedback

5.10.1.6.2.12 RowsAffected Property

Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.

Class

TCustomDASQL

Syntax

```
property RowsAffected: integer;
```

Remarks

Check RowsAffected to determine how many rows were inserted, updated, or deleted during the last query operation. If RowsAffected is -1, the query has not inserted, updated, or deleted any rows.

5.10.1.6.2.13 SQL Property

Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.

Class

TCustomDASQL

Syntax

```
property SQL: TStrings;
```

Remarks

Use the SQL property to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called. At design time the SQL property can be edited by invoking the String List editor in Object Inspector.

See Also

- FinalSQL
- TCustomDASQL.Execute

5.10.1.6.3 Methods

Methods of the **TCustomDASQL** class.

For a complete list of the **TCustomDASQL** class members, see the TCustomDASQL Members topic.

Public

Name	Description
BreakExec	Breaks execution of an SQL satatement on the server.

Execute	Overloaded. Executes a SQL statement on the server.
Executing	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro	Finds a macro with the specified name.
FindParam	Finds a parameter with the specified name.
MacroByName	Finds a macro with the specified name.
ParamByName	Finds a parameter with the specified name.
Prepare	Allocates, opens, and parses cursor for a query.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting	Waits until TCustomDASQL executes a SQL statement.

See Also

- TCustomDASQL Class
- TCustomDASQL Class Members

5.10.1.6.3.1 BreakExec Method

Breaks execution of an SQL satatement on the server.

Class

TCustomDASQL

Syntax

```
procedure BreakExec;
```

Remarks

Call the BreakExec method to break execution of an SQL statement on the server. It makes sense to call BreakExec only from another thread. Useful when NonBlocking is True.

See Also

- TCustomDASQL.Execute
- TCustomDADataSet.BreakExec

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.2 Execute Method

Executes a SQL statement on the server.

Class

TCustomDASQL

Overload List

Name	Description
Execute	Executes a SQL statement on the server.
Execute(Iters: integer; Offset: integer)	Used to perform Batch operations .

 Request Support DAC Forum Provide Feedback

Executes a SQL statement on the server.

Class

TCustomDASQL

Syntax

```
procedure Execute; overload; virtual;
```

Remarks

Call the Execute method to execute a SQL statement on the server. If the SQL statement has OUT parameters, use the TCustomDASQL.ParamByName method or the TCustomDASQL.Params property to get their values. Iters argument specifies the number of times this statement is executed for the DML array operations.

 Request Support DAC Forum Provide Feedback

Used to perform Batch operations .

Class

TCustomDASQL

Syntax

```
procedure Execute(Iters: integer; Offset: integer = 0); overload;
virtual;
```

Parameters

Iters

Specifies the number of inserted rows.

Offset

Points the array element, which the Batch operation starts from. 0 by default.

Remarks

The Execute method executes the specified batch SQL query. See the Batch operations article for samples.

See Also

- Batch operations

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.3 Executing Method

Checks whether TCustomDASQL still executes a SQL statement.

Class

TCustomDASQL

Syntax

```
function Executing: boolean;
```

Return Value

True, if a SQL statement is still being executed by TCustomDASQL.

Remarks

Check Executing to find out whether TCustomDASQL still executes a SQL statement. The Executing method is used for nonblocking execution.

Request Support DAC Forum Provide Feedback

5.10.1.6.3.4 FindMacro Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

```
function FindMacro(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

See Also

- TMacro

- Macros
- MacroByName

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.5 FindParam Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

```
function FindParam(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name to search for.

Return Value

a TDAParm object, if a parameter with the specified name has been found. If it has not, returns nil.

Remarks

Call the FindParam method to find a parameter with the specified name in a dataset.

See Also

- ParamByName

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.6 MacroByName Method

Finds a macro with the specified name.

Class

TCustomDASQL

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds the name of a macro to search for.

Return Value

TMacro object if a match is found.

Remarks

Call the MacroByName method to find a macro with the specified name. If a match is found, MacroByName returns the macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the TMacros.Items property to avoid depending on the order of the items.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

See Also

- TMacro
- Macros
- FindMacro

5.10.1.6.3.7 ParamByName Method

Finds a parameter with the specified name.

Class

TCustomDASQL

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the name of the parameter to search for.

Return Value

a TDAParam object, if a match was found. Otherwise, an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the specified name. If no parameter with the specified name found, an exception is raised.

Example

```
OraSQL.Execute;
Edit1.Text := OraSQL.ParamsByName('Contact').AsString;
```

See Also

- FindParam

5.10.1.6.3.8 Prepare Method

Allocates, opens, and parses cursor for a query.

Class

TCustomDASQL

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate, open, and parse cursor for a query. Calling Prepare before executing a query improves application performance.

TCustomDADataSet automatically prepares a query if it is executed without being prepared first. After execution, TCustomDADataSet unprepares the query. When a query is executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepared
- UnPrepare

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.9 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

TCustomDASQL

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free resources allocated for a previously prepared query on the server and client sides.

See Also

- Prepare

 Request Support DAC Forum Provide Feedback

5.10.1.6.3.10 WaitExecuting Method

Waits until TCustomDASQL executes a SQL statement.

Class

TCustomDASQL

Syntax

```
function WaitExecuting(TimeOut: integer = 0): boolean;
```

Parameters

TimeOut

Holds the time in seconds to wait while TCustomDASQL executes the SQL statement. Zero means infinite time.

Return Value

True, if the execution of a SQL statement was completed in the preset time.

Remarks

Call the WaitExecuting method to wait until TCustomDASQL executes a SQL statement. Use the WaitExecuting method for nonblocking execution.

See Also

- Executing

5.10.1.6.4 Events

Events of the **TCustomDASQL** class.

For a complete list of the **TCustomDASQL** class members, see the TCustomDASQL Members topic.

Public

Name	Description
AfterExecute	Occurs after a SQL statement has been executed.

See Also

- TCustomDASQL Class
- TCustomDASQL Class Members

5.10.1.6.4.1 AfterExecute Event

Occurs after a SQL statement has been executed.

Class

TCustomDASQL

Syntax

```
property AfterExecute: TAfterExecuteEvent;
```

Remarks

Occurs after a SQL statement has been executed. This event may be used for descendant components which use multithreaded environment.

See Also

- TCustomDASQL.Execute

Request Support DAC Forum Provide Feedback

5.10.1.7 TCustomDAUpdateSQL Class

A base class for components that provide DML statements for more flexible control over data modifications.

For a list of all members of this type, see TCustomDAUpdateSQL members.

Unit

DBAccess

Syntax

```
TCustomDAUpdateSQL = class(TComponent);
```

Remarks

TCustomDAUpdateSQL is a base class for components that provide DML statements for

more flexible control over data modifications. Besides providing BDE compatibility, this component allows to associate a separate component for each update command.

See Also

- TOraDataSet.UpdateObject

Request Support DAC Forum Provide Feedback

5.10.1.7.1 Members

TCustomDAUpdateSQL class overview.

Properties

Name	Description
DataSet	Used to hold a reference to the TCustomDADataSet object that is being updated.
DeleteObject	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL	Used when deleting a record.
InsertObject	Provides ability to perform advanced adjustment of insert operations.
InsertSQL	Used when inserting a record.
LockObject	Provides ability to perform advanced adjustment of lock operations.
LockSQL	Used to lock the current record.
ModifyObject	Provides ability to perform advanced adjustment of modify operations.
ModifySQL	Used when updating a record.
RefreshObject	Provides ability to perform advanced adjustment of refresh operations.

RefreshSQL	Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.
SQL	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Methods

Name	Description
Apply	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL	Executes a SQL statement.

Request Support DAC Forum Provide Feedback

5.10.1.7.2 Properties

Properties of the **TCustomDAUpdateSQL** class.

For a complete list of the **TCustomDAUpdateSQL** class members, see the TCustomDAUpdateSQL Members topic.

Public

Name	Description
DataSet	Used to hold a reference to the TCustomDADataSet object that is being updated.
SQL	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Published

Name	Description
DeleteObject	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL	Used when deleting a record.
InsertObject	Provides ability to perform advanced adjustment of insert operations.
InsertSQL	Used when inserting a record.
LockObject	Provides ability to perform advanced adjustment of lock operations.
LockSQL	Used to lock the current record.
ModifyObject	Provides ability to perform advanced adjustment of modify operations.
ModifySQL	Used when updating a record.
RefreshObject	Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL	Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

See Also

- TCustomDAUpdateSQL Class
- TCustomDAUpdateSQL Class Members

5.10.1.7.2.1 DataSet Property

Used to hold a reference to the TCustomDADataSet object that is being updated.

Class

TCustomDAUpdateSQL

Syntax

```
property DataSet: TCustomDADataSet;
```

Remarks

The DataSet property holds a reference to the TCustomDADataSet object that is being updated. Generally it is not used directly.

 Request Support DAC Forum Provide Feedback

5.10.1.7.2.2 DeleteObject Property

Provides ability to perform advanced adjustment of the delete operations.

Class

TCustomDAUpdateSQL

Syntax

```
property DeleteObject: TComponent;
```

Remarks

Assign SQL component or a TOraDataSet descendant to this property to perform advanced adjustment of the delete operations. In some cases this can give some additional performance. Use the same principle to set the SQL property of an object as for setting the DeleteSQL property.

See Also

- DeleteSQL

 Request Support DAC Forum Provide Feedback

Reserved.

5.10.1.7.2.3 DeleteSQL Property

Used when deleting a record.

Class

TCustomDAUpdateSQL

Syntax

```
property DeleteSQL: TStrings;
```

Remarks

Set the DeleteSQL property to a DELETE statement to use when deleting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

Request Support DAC Forum Provide Feedback

5.10.1.7.2.4 InsertObject Property

Provides ability to perform advanced adjustment of insert operations.

Class

TCustomDAUpdateSQL

Syntax

```
property InsertObject: TComponent;
```

Remarks

Assign SQL component or TOraDataSet descendant to this property to perform advanced adjustment of insert operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the InsertSQL property.

See Also

- InsertSQL

5.10.1.7.2.5 InsertSQL Property

Used when inserting a record.

Class

TCustomDAUpdateSQL

Syntax

```
property InsertSQL: TStrings;
```

Remarks

Set the InsertSQL property to an INSERT INTO statement to use when inserting a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

5.10.1.7.2.6 LockObject Property

Provides ability to perform advanced adjustment of lock operations.

Class

TCustomDAUpdateSQL

Syntax

```
property LockObject: TComponent;
```

Remarks

Assign a SQL component or TOraDataSet descendant to this property to perform advanced adjustment of lock operations. In some cases that can give some additional performance. Set the SQL property of an object in the same way as used for the LockSQL property.

See Also

- LockSQL

 Request Support DAC Forum Provide Feedback

5.10.1.7.2.7 LockSQL Property

Used to lock the current record.

Class

TCustomDAUpdateSQL

Syntax

```
property LockSQL: TStrings;
```

Remarks

Use the LockSQL property to lock the current record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

 Request Support DAC Forum Provide Feedback

5.10.1.7.2.8 ModifyObject Property

Provides ability to perform advanced adjustment of modify operations.

Class

TCustomDAUpdateSQL

Syntax

```
property ModifyObject: TComponent;
```

Remarks

Assign a SQL component or TOraDataSet descendant to this property to perform advanced adjustment of modify operations. In some cases this can give some additional performance. Set the SQL property of the object in the same way as used for the ModifySQL property.

See Also

- ModifySQL

5.10.1.7.2.9 ModifySQL Property

Used when updating a record.

Class

TCustomDAUpdateSQL

Syntax

```
property ModifySQL: TStrings;
```

Remarks

Set ModifySQL to an UPDATE statement to use when updating a record. Statements can be parameterized queries with parameter names corresponding to the dataset field names.

5.10.1.7.2.10 RefreshObject Property

Provides ability to perform advanced adjustment of refresh operations.

Class

TCustomDAUpdateSQL

Syntax

```
property RefreshObject: TComponent;
```

Remarks

Assign a SQL component or TOraDataSet descendant to this property to perform advanced adjustment of refresh operations. In some cases that can give some additional performance. Set the SQL property of the object in the same way as used for the RefreshSQL property.

See Also

- RefreshSQL

5.10.1.7.2.11 RefreshSQL Property

Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.

Class

TCustomDAUpdateSQL

Syntax

```
property RefreshSQL: TStrings;
```

Remarks

Use the RefreshSQL property to specify a SQL statement that will be used for refreshing the current record by the TCustomDADataSet.RefreshRecord procedure.

You can assign to SQLRefresh a WHERE clause only. In such a case it is added to SELECT defined by the SQL property by TCustomDADataSet.AddWhere.

To create a RefreshSQL statement at design time, use the query statements editor.

See Also

- TCustomDADataSet.RefreshRecord

5.10.1.7.2.12 SQL Property(Indexer)

Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Class

TCustomDAUpdateSQL

Syntax

```
property SQL[UpdateKind: TUpdateKind]: TStrings;
```

Parameters

UpdateKind

Specifies which of update SQL statements to return.

Remarks

Returns a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties, depending on the value of the UpdateKind index.

 Request Support DAC Forum Provide Feedback

5.10.1.7.3 Methods

Methods of the **TCustomDAUpdateSQL** class.

For a complete list of the **TCustomDAUpdateSQL** class members, see the TCustomDAUpdateSQL Members topic.

Public

Name	Description
Apply	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL	Executes a SQL statement.

See Also

- TCustomDAUpdateSQL Class
- TCustomDAUpdateSQL Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.7.3.1 Apply Method

Sets parameters for a SQL statement and executes it to update a record.

Class

TCustomDAUpdateSQL

Syntax

```
procedure Apply(UpdateKind: TUpdateKind); virtual;
```

Parameters

UpdateKind

Specifies which of update SQL statements to execute.

Remarks

Call the Apply method to set parameters for a SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute.

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

Note: If a SQL statement does not contain parameters, it is more efficient to call ExecSQL instead of Apply.

See Also

- ExecSQL

5.10.1.7.3.2 ExecSQL Method

Executes a SQL statement.

Class

TCustomDAUpdateSQL

Syntax

```
procedure ExecSQL(UpdateKind: TUpdateKind);
```

Parameters

UpdateKind

Specifies the kind of update statement to be executed.

Remarks

Call the ExecSQL method to execute a SQL statement, necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute.

ExecSQL is primarily intended for manually executing update statements from the OnUpdateRecord event handler.

Note: To both bind parameters and execute a statement, call Apply.

See Also

- Apply

 Request Support DAC Forum Provide Feedback

5.10.1.8 TDACondition Class

Represents a condition from the TDAConditions list.

For a list of all members of this type, see TDACondition members.

Unit

DBAccess

Syntax

```
TDACondition = class(TCollectionItem);
```

Remarks

Manipulate conditions using TDAConditions.

See Also

- TDAConditions

 Request Support DAC Forum Provide Feedback

5.10.1.8.1 Members

TDACondition class overview.

Properties

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

Methods

Name	Description
Disable	Disables the condition
Enable	Enables the condition

Request Support DAC Forum Provide Feedback

5.10.1.8.2 Properties

Properties of the **TDACondition** class.

For a complete list of the **TDACondition** class members, see the TDACondition Members topic.

Published

Name	Description
Enabled	Indicates whether the condition is enabled or not
Name	The name of the condition
Value	The value of the condition

See Also

- TDACondition Class
- TDACondition Class Members

5.10.1.8.2.1 Enabled Property

Indicates whether the condition is enabled or not

Class

TDACondition

Syntax

```
property Enabled: Boolean default True;
```


5.10.1.8.2.2 Name Property

The name of the condition

Class

TDACondition

Syntax

```
property Name: string;
```


5.10.1.8.2.3 Value Property

The value of the condition

Class

TDACondition

Syntax

```
property Value: string;
```

 Request Support DAC Forum Provide Feedback

5.10.1.8.3 Methods

Methods of the **TDACondition** class.

For a complete list of the **TDACondition** class members, see the TDACondition Members topic.

Public

Name	Description
Disable	Disables the condition
Enable	Enables the condition

See Also

- TDACondition Class
- TDACondition Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.8.3.1 Disable Method

Disables the condition

Class

TDACondition

Syntax

```
procedure Disable;
```

 Request Support DAC Forum Provide Feedback

5.10.1.8.3.2 Enable Method

Enables the condition

Class

TDACondition

Syntax

```
procedure Enable;
```

 Request Support DAC Forum Provide Feedback

5.10.1.9 TDAConditions Class

Holds a collection of TDACondition objects.

For a list of all members of this type, see TDAConditions members.

Unit

DBAccess

Syntax

```
TDAConditions = class(TCollection);
```

Remarks

The given example code

```
UniTable1.Conditions.Add('1','JOB="MANAGER"');
UniTable1.Conditions.Add('2','SAL>2500');
UniTable1.Conditions.Enable;
UniTable1.Open;
```

will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

 Request Support DAC Forum Provide Feedback

5.10.1.9.1 Members

TDAConditions class overview.

Properties

Name	Description
Condition	Used to iterate through all the conditions.
Enabled	Indicates whether the condition is enabled
Items	Used to iterate through all conditions.
Text	The property returns condition names and values as CONDITION_NAME=CONDITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

Methods

Name	Description
Add	Overloaded. Adds a condition to the WHERE clause of the query.
Delete	Deletes the condition
Disable	Disables the condition
Enable	Enables the condition
Find	Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.
Get	Retrieving a TDACondition object by its name. If found, the TDACondition object is

	returned, otherwise - an exception is raised.
IndexOf	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
Remove	Removes the condition

 Request Support DAC Forum Provide Feedback

5.10.1.9.2 Properties

Properties of the **TDAConditions** class.

For a complete list of the **TDAConditions** class members, see the TDAConditions Members topic.

Public

Name	Description
Condition	Used to iterate through all the conditions.
Enabled	Indicates whether the condition is enabled
Items	Used to iterate through all conditions.
Text	The property returns condition names and values as CONDITION_NAME=CONDITION
WhereSQL	Returns the SQL WHERE condition added in the Conditions property.

See Also

- TDAConditions Class
- TDAConditions Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.9.2.1 Condition Property(Indexer)

Used to iterate through all the conditions.

Class

TDACConditions

Syntax

```
property Condition[Index: Integer]: TDACCondition;
```

Parameters

Index

5.10.1.9.2.2 Enabled Property

Indicates whether the condition is enabled

Class

TDACConditions

Syntax

```
property Enabled: Boolean;
```


5.10.1.9.2.3 Items Property(Indexer)

Used to iterate through all conditions.

Class

TDACConditions

Syntax

```
property Items[Index: Integer]: TDACondition; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all conditions. Index identifies the index in the range 0..Count - 1. Items can reference a particular condition by its index, but the Condition property is preferred in order to avoid depending on the order of the conditions.

Request Support DAC Forum Provide Feedback

5.10.1.9.2.4 Text Property

The property returns condition names and values as CONDITION_NAME=CONDITION

Class

TDAConditions

Syntax

```
property Text: string;
```


Request Support DAC Forum Provide Feedback

5.10.1.9.2.5 WhereSQL Property

Returns the SQL WHERE condition added in the Conditions property.

Class

TDAConditions

Syntax

```
property WhereSQL: string;
```


Request Support DAC Forum Provide Feedback

5.10.1.9.3 Methods

Methods of the **TDAConditions** class.

For a complete list of the **TDAConditions** class members, see the TDAConditions Members topic.

Public

Name	Description
Add	Overloaded. Adds a condition to the WHERE clause of the query.
Delete	Deletes the condition
Disable	Disables the condition
Enable	Enables the condition
Find	Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.
Get	Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.
IndexOf	Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.
Remove	Removes the condition

See Also

- TDAConditions Class
- TDAConditions Class Members

5.10.1.9.3.1 Add Method

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Overload List

Name	Description
Add(**const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.
Add(**const** Name: **string**; **const** Value: **string**; Enabled: Boolean)	Adds a condition to the WHERE clause of the query.

Request Support DAC Forum Provide Feedback

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```
function Add(const Value: string; Enabled: Boolean = True):
TDACondition; overload;
```

Parameters

Value

The value of the condition

Enabled

Indicates that the condition is enabled

Remarks

If you want then to access the condition, you should use Add and its name in the Name parameter.

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
```

```
and
(SAL<2500)
```

 Request Support DAC Forum Provide Feedback

Adds a condition to the WHERE clause of the query.

Class

TDAConditions

Syntax

```
function Add(const Name: string; const Value: string; Enabled:
Boolean = True): TDACondition; overload;
```

Parameters

Name

Sets the name of the condition

Value

The value of the condition

Enabled

Indicates that the condition is enabled

Remarks

The given example code will return the following SQL:

```
SELECT * FROM EMP
WHERE (JOB="MANAGER")
and
(SAL<2500)
```

 Request Support DAC Forum Provide Feedback

5.10.1.9.3.2 Delete Method

Deletes the condition

Class

TDAConditions

Syntax

```
procedure Delete(Index: integer);
```

Parameters

Index

Index of the condition

 Request Support DAC Forum Provide Feedback

5.10.1.9.3.3 Disable Method

Disables the condition

Class

TDAConditions

Syntax

```
procedure Disable;
```

 Request Support DAC Forum Provide Feedback

5.10.1.9.3.4 Enable Method

Enables the condition

Class

TDAConditions

Syntax

```
procedure Enable;
```

 Request Support DAC Forum Provide Feedback

5.10.1.9.3.5 Find Method

Search for TDACondition (the condition) by its name. If found, the TDACondition object is returned, otherwise - nil.

Class

TDAConditions

Syntax

```
function Find(const Name: string): TDACondition;
```

Parameters

Name

Request Support DAC Forum Provide Feedback

5.10.1.9.3.6 Get Method

Retrieving a TDACondition object by its name. If found, the TDACondition object is returned, otherwise - an exception is raised.

Class

TDAConditions

Syntax

```
function Get(const Name: string): TDACondition;
```

Parameters

Name

Request Support DAC Forum Provide Feedback

5.10.1.9.3.7 IndexOf Method

Retrieving condition index by its name. If found, this condition index is returned, otherwise - the method returns -1.

Class

TDAConditions

Syntax

```
function IndexOf(const Name: string): Integer;
```

Parameters

Name

5.10.1.9.3.8 Remove Method

Removes the condition

Class

TDAConditions

Syntax

```
procedure Remove(const Name: string);
```

Parameters

Name

Specifies the name of the removed condition

5.10.1.10 TDAConnectionOptions Class

This class allows setting up the behaviour of the TDAConnection class.

For a list of all members of this type, see TDAConnectionOptions members.

Unit

DBAccess

Syntax

```
TDAConnectionOptions = class(TPersistent);
```


5.10.1.10.1 Members

TDAConnectionOptions class overview.

Properties

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.
DefaultSortType	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Request Support DAC Forum Provide Feedback

5.10.1.10.2 Properties

Properties of the **TDAConnectionOptions** class.

For a complete list of the **TDAConnectionOptions** class members, see the TDAConnectionOptions Members topic.

Public

Name	Description
DefaultSortType	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect	Specifies whether to allow or not implicit connection opening.

See Also

- TDAConnectionOptions Class
- TDAConnectionOptions Class Members

5.10.1.10.2.1 Allow ImplicitConnect Property

Specifies whether to allow or not implicit connection opening.

Class

TDAConnectionOptions

Syntax

```
property AllowImplicitConnect: boolean default True;
```

Remarks

Use the AllowImplicitConnect property to specify whether allow or not implicit connection opening.

If a closed connection has AllowImplicitConnect set to True and a dataset that uses the connection is opened, the connection is opened implicitly to allow opening the dataset.

If a closed connection has AllowImplicitConnect set to False and a dataset that uses the connection is opened, an exception is raised.

The default value is True.

 Request Support DAC Forum Provide Feedback

5.10.1.10.2.2 DefaultSortType Property

Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

Class

TDAConnectionOptions

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks

Use the DefaultSortType property to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

5.10.1.10.2.3 DisconnectedMode Property

Used to open a connection only when needed for performing a server call and closes after performing the operation.

Class

TDAConnectionOptions

Syntax

```
property DisconnectedMode: boolean default False;
```

Remarks

If True, connection opens only when needed for performing a server call and closes after performing the operation. Datasets remain opened when connection closes. May be useful to save server resources and operate in unstable or expensive network. Drawback of using disconnect mode is that each connection establishing requires some time for authorization. If connection is often closed and opened it can slow down the application work. See the Disconnected Mode topic for more information.

5.10.1.10.2.4 KeepDesignConnected Property

Used to prevent an application from establishing a connection at the time of startup.

Class

TDAConnectionOptions

Syntax

```
property KeepDesignConnected: boolean default True;
```

Remarks

At the time of startup prevents application from establishing a connection even if the Connected property was set to True at design-time. Set KeepDesignConnected to False to initialize the connected property to False, even if it was True at design-time.

 Request Support DAC Forum Provide Feedback

5.10.1.10.2.5 LocalFailover Property

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Class

TDAConnectionOptions

Syntax

```
property LocalFailover: boolean default False;
```

Remarks

If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks. Read the Working in an Unstable Network topic for more information about using failover.

 Request Support DAC Forum Provide Feedback

5.10.1.11 TDAConnectionSSLOptions Class

This class is used to set up the SSL options.

For a list of all members of this type, see TDAConnectionSSLOptions members.

Unit

DBAccess

Syntax

```
TDAConnectionSSLOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.10.1.11.1 Members

TDAConnectionSSLOptions class overview.

Properties

Name	Description
CACert	Holds the path to the certificate authority file.
Cert	Holds the path to the client certificate.
CipherList	Holds the list of allowed SSL ciphers.
Key	Holds the path to the private client key.

 Request Support DAC Forum Provide Feedback

5.10.1.11.2 Properties

Properties of the **TDAConnectionSSLOptions** class.

For a complete list of the **TDAConnectionSSLOptions** class members, see the TDAConnectionSSLOptions Members topic.

Published

Name	Description
CACert	Holds the path to the certificate authority file.
Cert	Holds the path to the client certificate.
CipherList	Holds the list of allowed SSL ciphers.

Key	Holds the path to the private client key.

See Also

- TDAConnectionSSLOptions Class
- TDAConnectionSSLOptions Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.11.2.1 CACert Property

Holds the path to the certificate authority file.

Class

TDAConnectionSSLOptions

Syntax

```
property CACert: string;
```

Remarks

Use the CACert property to specify the path to the certificate authority file.

 Request Support DAC Forum Provide Feedback

5.10.1.11.2.2 Cert Property

Holds the path to the client certificate.

Class

TDAConnectionSSLOptions

Syntax

```
property Cert: string;
```

Remarks

Use the Cert property to specify the path to the client certificate.

 Request Support DAC Forum Provide Feedback

5.10.1.11.2.3 CipherList Property

Holds the list of allowed SSL ciphers.

Class

TDAConnectionSSLOptions

Syntax

```
property CipherList: string;
```

Remarks

Use the CipherList property to specify the list of allowed SSL ciphers.

 Request Support DAC Forum Provide Feedback

5.10.1.11.2.4 Key Property

Holds the path to the private client key.

Class

TDAConnectionSSLOptions

Syntax

```
property Key: string;
```

Remarks

Use the Key property to specify the path to the private client key.

 Request Support DAC Forum Provide Feedback

5.10.1.12 TDADataSetOptions Class

This class allows setting up the behaviour of the TDADataSet class.

For a list of all members of this type, see TDADataSetOptions members.

Unit

DBAccess

Syntax

```
TDADataSetOptions = class(TPersistent);
```


Request Support DAC Forum Provide Feedback

5.10.1.12.1 Members

TDADataSetOptions class overview.

Properties

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill

	the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object

	names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support DAC Forum Provide Feedback

5.10.1.12.2 Properties

Properties of the **TDADataSetOptions** class.

For a complete list of the **TDADataSetOptions** class members, see the TDADataSetOptions Members topic.

Public

Name	Description
AutoPrepare	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used to request default values/expressions from the server and assign them to the DefaultExpression property.
DetailDelay	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail	Used for TCustomDADataSet to use

	local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams	Used to return the new value of fields to dataset after insert or update.

SetFieldsReadOnly	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

See Also

- TDADataSetOptions Class
- TDADataSetOptions Class Members

5.10.1.12.2.1 AutoPrepare Property

Used to execute automatic TCustomDADataSet.Prepare on the query execution.

Class

TDADataSetOptions

Syntax

```
property AutoPrepare: boolean default False;
```

Remarks

Use the AutoPrepare property to execute automatic TCustomDADataSet.Prepare on the query execution. Makes sense for cases when a query will be executed several times, for example, in Master/Detail relationships.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.2 CacheCalcFields Property

Used to enable caching of the TField.Calculated and TField.Lookup fields.

Class

TDADataSetOptions

Syntax

```
property CacheCalcFields: boolean default False;
```

Remarks

Use the CacheCalcFields property to enable caching of the TField.Calculated and TField.Lookup fields. It can be useful for reducing CPU usage for calculated fields. Using caching of calculated and lookup fields increases memory usage on the client side.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.3 CompressBlobMode Property

Used to store values of the BLOB fields in compressed form.

Class

TDADataSetOptions

Syntax

```
property CompressBlobMode: TCompressBlobMode default cbNone;
```

Remarks

Use the CompressBlobMode property to store values of the BLOB fields in compressed form. Add the MemData unit to uses list to use this option. Compression rate greatly depends on stored data, for example, usually graphic data compresses badly unlike text.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.4 DefaultValues Property

Used to request default values/expressions from the server and assign them to the DefaultExpression property.

Class

TDADataSetOptions

Syntax

```
property DefaultValues: boolean default False;
```

Remarks

If True, the default values/expressions are requested from the server and assigned to the DefaultExpression property of TField objects replacing already existent values.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.5 DetailDelay Property

Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.

Class

TDADataSetOptions

Syntax

```
property DetailDelay: integer default 0;
```

Remarks

Use the DetailDelay property to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset. If DetailDelay is 0 (the default value) then refreshing of detail dataset occurs immediately. The DetailDelay option should be used for detail dataset.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.6 FieldsOrigin Property

Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.

Class

TDADataSetOptions

Syntax

```
property FieldsOrigin: boolean;
```

Remarks

If True, TCustomDADataSet fills the Origin property of the TField objects by appropriate value when opening a dataset.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.7 FlatBuffers Property

Used to control how a dataset treats data of the ftString and ftVarBytes fields.

Class

TDADataSetOptions

Syntax

```
property FlatBuffers: boolean default False;
```

Remarks

Use the FlatBuffers property to control how a dataset treats data of the ftString and ftVarBytes

fields. When set to True, all data fetched from the server is stored in record pdata without unused tails.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.8 InsertAllSetFields Property

Used to include all set dataset fields in the generated INSERT statement

Class

TDADataSetOptions

Syntax

```
property InsertAllSetFields: boolean default False;
```

Remarks

If True, all set dataset fields, including those set to NULL explicitly, will be included in the generated INSERT statements. Otherwise, only set fields containing not NULL values will be included to the generated INSERT statement.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.9 LocalMasterDetail Property

Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.

Class

TDADataSetOptions

Syntax

```
property LocalMasterDetail: boolean default False;
```

Remarks

If True, for detail dataset in master-detail relationship TCustomDADataSet uses local filtering

for establishing master/detail relationship and does not refer to the server. Otherwise detail dataset performs query each time a record is selected in master dataset. This option is useful for reducing server calls number, server resources economy. It can be useful for slow connection. The TMemDataSet.CachedUpdates mode can be used for detail dataset only when this option is set to true. Setting the LocalMasterDetail option to True is not recommended when detail table contains too many rows, because when it is set to False, only records that correspond to the current record in master dataset are fetched.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.10 LongStrings Property

Used to represent string fields with the length that is greater than 255 as TStringField.

Class

TDADataSetOptions

Syntax

```
property LongStrings: boolean default True;
```

Remarks

Use the LongStrings property to represent string fields with the length that is greater than 255 as TStringField, not as TMemoField.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.11 MasterFieldsNullable Property

Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).

Class

TDADataSetOptions

Syntax

```
property MasterFieldsNullable: boolean default False;
```


5.10.1.12.2.12 NumberRange Property

Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

Class

TDADataSetOptions

Syntax

```
property NumberRange: boolean default False;
```

Remarks

Use the NumberRange property to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.

5.10.1.12.2.13 QueryRecCount Property

Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.

Class

TDADataSetOptions

Syntax

```
property QueryRecCount: boolean default False;
```

Remarks

If True, and the FetchAll property is False, TCustomDADataSet performs additional query to get the record count for this SELECT, so the RecordCount property reflects the actual

number of records. Does not have any effect if the FetchAll property is True.

5.10.1.12.2.14 QuoteNames Property

Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.

Class

TDADataSetOptions

Syntax

```
property QuoteNames: boolean default False;
```

Remarks

If True, TCustomDADataSet quotes all database object names in autogenerated SQL statements such as update SQL.

5.10.1.12.2.15 RemoveOnRefresh Property

Used for a dataset to locally remove a record that can not be found on the server.

Class

TDADataSetOptions

Syntax

```
property RemoveOnRefresh: boolean default True;
```

Remarks

When the RefreshRecord procedure can't find necessary record on the server and RemoveOnRefresh is set to True, dataset removes the record locally. Usually RefreshRecord can't find necessary record when someone else dropped the record or

changed the key value of it.

This option makes sense only if the StrictUpdate option is set to False. If the StrictUpdate option is True, error will be generated regardless of the RemoveOnRefresh option value.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.16 RequiredFields Property

Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.

Class

TDADataSetOptions

Syntax

```
property RequiredFields: boolean default True;
```

Remarks

If True, TCustomDADataSet sets the Required property of the TField objects for the NOT NULL fields. It is useful when table has a trigger which updates the NOT NULL fields.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.17 ReturnParams Property

Used to return the new value of fields to dataset after insert or update.

Class

TDADataSetOptions

Syntax

```
property ReturnParams: boolean default False;
```

Remarks

Use the ReturnParams property to return the new value of fields to dataset after insert or

update. The actual value of field after insert or update may be different from the value stored in the local memory if the table has a trigger. When ReturnParams is True, OUT parameters of the SQLInsert and SQLUpdate statements is assigned to the corresponding fields.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.18 SetFieldsReadOnly Property

Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.

Class

TDADataSetOptions

Syntax

```
property SetFieldsReadOnly: boolean default True;
```

Remarks

If True, dataset sets the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated. Set this option for datasets that use automatic generation of the update SQL statements only.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.19 StrictUpdate Property

Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.

Class

TDADataSetOptions

Syntax

```
property StrictUpdate: boolean default True;
```

Remarks

If True, TCustomDADataSet raises an exception when the number of updated or deleted records is not equal 1. Setting this option also causes the exception if the RefreshRecord procedure returns more than one record. The exception does not occur when you execute SQL query, that doesn't return resultset.

Note: There can be problems if this option is set to True and triggers for UPDATE, DELETE, REFRESH commands that are defined for the table. So it is recommended to disable (set to False) this option with triggers.

TrimFixedChar specifies whether to discard all trailing spaces in the string fields of a dataset.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.20 TrimFixedChar Property

Specifies whether to discard all trailing spaces in the string fields of a dataset.

Class

TDADataSetOptions

Syntax

```
property TrimFixedChar: boolean default True;
```

Remarks

Specifies whether to discard all trailing spaces in the string fields of a dataset.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.21 UpdateAllFields Property

Used to include all dataset fields in the generated UPDATE and INSERT statements.

Class

TDADataSetOptions

Syntax

```
property UpdateAllFields: boolean default False;
```

Remarks

If True, all dataset fields will be included in the generated UPDATE and INSERT statements. Unspecified fields will have NULL value in the INSERT statements. Otherwise, only updated fields will be included to the generated update statements.

 Request Support DAC Forum Provide Feedback

5.10.1.12.2.22 UpdateBatchSize Property

Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Class

TDADataSetOptions

Syntax

```
property UpdateBatchSize: Integer default 1;
```

Remarks

Use the UpdateBatchSize property to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch. Takes effect only when updating dataset in the TMemDataSet.CachedUpdates mode. The default value is 1.

 Request Support DAC Forum Provide Feedback

5.10.1.13 TDAEncryption Class

Used to specify the options of the data encryption in a dataset.

For a list of all members of this type, see TDAEncryption members.

Unit

DBAccess

Syntax

```
TDAEncryption = class(TPersistent);
```

Remarks

Set the properties of Encryption to specify the options of the data encryption in a dataset.

 Request Support DAC Forum Provide Feedback

5.10.1.13.1 Members

TDAEncryption class overview.

Properties

Name	Description
Encryptor	Used to specify the encryptor class that will perform the data encryption.
Fields	Used to set field names for which encryption will be performed.

 Request Support DAC Forum Provide Feedback

5.10.1.13.2 Properties

Properties of the **TDAEncryption** class.

For a complete list of the **TDAEncryption** class members, see the TDAEncryption Members topic.

Public

Name	Description
Encryptor	Used to specify the encryptor class that will perform the data encryption.

Published

Name	Description
Fields	Used to set field names for which encryption will be performed.

See Also

- TDAEncryption Class
- TDAEncryption Class Members

Request Support DAC Forum Provide Feedback

5.10.1.13.2.1 Encryptor Property

Used to specify the encryptor class that will perform the data encryption.

Class

TDAEncryption

Syntax

```
property Encryptor: TCREncryptor;
```

Remarks

Use the Encryptor property to specify the encryptor class that will perform the data encryption.

Request Support DAC Forum Provide Feedback

5.10.1.13.2.2 Fields Property

Used to set field names for which encryption will be performed.

Class

TDAEncryption

Syntax

```
property Fields: string;
```

Remarks

Used to set field names for which encryption will be performed. Field names must be separated by semicolons.

 Request Support DAC Forum Provide Feedback

5.10.1.14 TDAMapRule Class

Class that formes rules for Data Type Mapping.

For a list of all members of this type, see TDAMapRule members.

Unit

DBAccess

Syntax

```
TDAMapRule = class(TMapRule);
```

Remarks

Using properties of this class, it is possible to change parameter values of the specified rules from the TDAMapRules set.

Inheritance Hierarchy

TMapRule

TDAMapRule

 Request Support DAC Forum Provide Feedback

5.10.1.14.1 Members

TDAMapRule class overview.

Properties

Name	Description
DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

Request Support DAC Forum Provide Feedback

5.10.1.14.2 Properties

Properties of the **TDAMapRule** class.

For a complete list of the **TDAMapRule** class members, see the TDAMapRule Members topic.

Published

Name	Description

DBLengthMax	Maximum DB field length, until which the rule is applied.
DBLengthMin	Minimum DB field length, starting from which the rule is applied.
DBScaleMax	Maximum DB field scale, until which the rule is applied to the specified DB field.
DBScaleMin	Minimum DB field Scale, starting from which the rule is applied to the specified DB field.
DBType	DB field type, that the rule is applied to.
FieldLength	The resultant field length in Delphi.
FieldName	DataSet field name, for which the rule is applied.
FieldScale	The resultant field Scale in Delphi.
FieldType	Delphi field type, that the specified DB type or DataSet field will be mapped to.
IgnoreErrors	Ignoring errors when converting data from DB to Delphi type.

See Also

- TDAMapRule Class
- TDAMapRule Class Members

Request Support DAC Forum Provide Feedback

5.10.1.14.2.1 DBLengthMax Property

Maximum DB field length, until which the rule is applied.

Class

TDAMapRule

Syntax

```
property DBLengthMax: Integer default rlAny;
```

Remarks

Setting maximum DB field length, until which the rule is applied to the specified DB field.

5.10.1.14.2.2 DBLengthMin Property

Minimum DB field length, starting from which the rule is applied.

Class

TDAMapRule

Syntax

```
property DBLengthMin: Integer default rlAny;
```

Remarks

Setting minimum DB field length, starting from which the rule is applied to the specified DB field.

5.10.1.14.2.3 DBScaleMax Property

Maximum DB field scale, until which the rule is applied to the specified DB field.

Class

TDAMapRule

Syntax

```
property DBScaleMax: Integer default rlAny;
```

Remarks

Setting maximum DB field scale, until which the rule is applied to the specified DB field.

 Request Support DAC Forum Provide Feedback

5.10.1.14.2.4 DBScaleMin Property

Minimum DB field Scale, starting from which the rule is applied to the specified DB field.

Class

TDAMapRule

Syntax

```
property DBScaleMin: Integer default rlAny;
```

Remarks

Setting minimum DB field Scale, starting from which the rule is applied to the specified DB field.

 Request Support DAC Forum Provide Feedback

5.10.1.14.2.5 DBType Property

DB field type, that the rule is applied to.

Class

TDAMapRule

Syntax

```
property DBType: Word default dtUnknown;
```

Remarks

Setting DB field type, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields of the specified type in all DataSets related to this Connection.

 Request Support DAC Forum Provide Feedback

5.10.1.14.2.6 FieldLength Property

The resultant field length in Delphi.

Class

TDAMapRule

Syntax

```
property FieldLength: Integer default rlAny;
```

Remarks

Setting the Delphi field length after conversion.

 Request Support DAC Forum Provide Feedback

5.10.1.14.2.7 FieldName Property

DataSet field name, for which the rule is applied.

Class

TDAMapRule

Syntax

```
property FieldName: string;
```

Remarks

Specifies the DataSet field name, that the rule is applied to. If the current rule is set for Connection, the rule will be applied to all fields with such name in DataSets related to this Connection.

 Request Support DAC Forum Provide Feedback

5.10.1.14.2.8 FieldScale Property

The resultant field Scale in Delphi.

Class

TDAMapRule

Syntax

```
property FieldScale: Integer default rlAny;
```

Remarks

Setting the Delphi field Scale after conversion.

Request Support DAC Forum Provide Feedback

5.10.1.14.2.9 FieldType Property

Delphi field type, that the specified DB type or DataSet field will be mapped to.

Class

TDAMapRule

Syntax

```
property FieldType: TFieldType stored IsFieldTypeStored default
ftUnknown;
```

Remarks

Setting Delphi field type, that the specified DB type or DataSet field will be mapped to.

Request Support DAC Forum Provide Feedback

5.10.1.14.2.10 IgnoreErrors Property

Ignoring errors when converting data from DB to Delphi type.

Class

TDAMapRule

Syntax

```
property IgnoreErrors: Boolean default False;
```

Remarks

Allows to ignore errors while data conversion in case if data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

 Request Support DAC Forum Provide Feedback

5.10.1.15 TDAMapRules Class

Used for adding rules for DataSet fields mapping with both identifying by field name and by field type and Delphi field types.

For a list of all members of this type, see TDAMapRules members.

Unit

DBAccess

Syntax

```
TDAMapRules = class(TMapRules);
```

Inheritance Hierarchy

TMapRules

TDAMapRules

 Request Support DAC Forum Provide Feedback

5.10.1.15.1 Members

TDAMapRules class overview.

Properties

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

Request Support DAC Forum Provide Feedback

5.10.1.15.2 Properties

Properties of the **TDAMapRules** class.

For a complete list of the **TDAMapRules** class members, see the TDAMapRules Members topic.

Published

Name	Description
IgnoreInvalidRules	Used to avoid raising exception on mapping rules that can't be applied.

See Also

- TDAMapRules Class
- TDAMapRules Class Members

Request Support DAC Forum Provide Feedback

5.10.1.15.2.1 IgnoreInvalidRules Property

Used to avoid raising exception on mapping rules that can't be applied.

Class

TDAMapRules

Syntax

```
property IgnoreInvalidRules: boolean default False;
```

Remarks

Allows to ignore errors (not to raise exception) during data conversion in case if the data or DB data format cannot be recorded to the specified Delphi field type. The default value is false.

Note: In order to ignore errors occurring during data conversion, use the TDAMapRule.IgnoreErrors property

See Also

- TDAMapRule.IgnoreErrors

Request Support DAC Forum Provide Feedback

5.10.1.16 TDAMetaData Class

A class for retrieving metainformation of the specified database objects in the form of dataset.

For a list of all members of this type, see TDAMetaData members.

Unit

DBAccess

Syntax

```
TDAMetaData = class(TMemDataSet);
```

Remarks

TDAMetaData is a TDataSet descendant standing for retrieving metainformation of the specified database objects in the form of dataset. First of all you need to specify which kind of metainformation you want to see. For this you need to assign the TDAMetaData.MetaDataKind property. Provide one or more conditions in the TDAMetaData.Restrictions property to diminish the size of the resultset and get only information you are interested in.

Use the TDAMetaData.GetMetaDataKinds method to get the full list of supported kinds of meta data. With the TDAMetaData.GetRestrictions method you can find out what restrictions are applicable to the specified MetaDataKind.

Example

The code below demonstrates how to get information about columns of the 'emp' table:

```
MetaData.Connection := Connection;
MetaData.MetaDataKind := 'Columns';
MetaData.Restrictions.Values['TABLE_NAME'] := 'Emp';
MetaData.Open;
```

Inheritance Hierarchy

TMemDataSet

TDAMetaData

See Also

- TDAMetaData.MetaDataKind
- TDAMetaData.Restrictions
- TDAMetaData.GetMetaDataKinds
- TDAMetaData.GetRestrictions

5.10.1.16.1 Members

TDAMetaData class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Connection	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.

LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.

EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.

SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

5.10.1.16.2 Properties

Properties of the **TDAMetaData** class.

For a complete list of the **TDAMetaData** class members, see the TDAMetaData Members topic.

Public

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
Connection	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TDAMetaData Class

- TDAMetaData Class Members

5.10.1.16.2.1 Connection Property

Used to specify a connection object to use to connect to a data store.

Class

TDAMetaData

Syntax

```
property Connection: TCustomDAConnection;
```

Remarks

Use the Connection property to specify a connection object to use to connect to a data store.

Set at design-time by selecting from the list of provided TCustomDAConnection or its descendant class objects.

At runtime, set the Connection property to reference an instanciated TCustomDAConnection object.

5.10.1.16.2.2 MetaDataKind Property

Used to specify which kind of metainformation to show.

Class

TDAMetaData

Syntax

```
property MetaDataKind: string;
```

Remarks

This string property specifies which kind of metainformation to show. The value of this property should be assigned before activating the component. If MetaDataKind equals to an empty string (the default value), the full value list that this property accepts will be shown.

They are described in the table below:

MetaDataKind	Description
Columns	show metainformation about columns of existing tables
Constraints	show metainformation about the constraints defined in the database
IndexColumns	show metainformation about indexed columns
Indexes	show metainformation about indexes in a database
MetaDataKinds	show the acceptable values of this property. You will get the same result if the MetadataKind property is an empty string
ProcedureParameters	show metainformation about parameters of existing procedures
Procedures	show metainformation about existing procedures
Restrictions	generates a dataset that describes which restrictions are applicable to each MetaDataKind
Tables	show metainformation about existing tables
Databases	show metainformation about existing databases

If you provide a value that equals neither of the values described in the table, an error will be raised.

See Also

- Restrictions

5.10.1.16.2.3 Restrictions Property

Used to provide one or more conditions restricting the list of objects to be described.

Class

TDAMetaData

Syntax

```
property Restrictions: TStrings;
```

Remarks

Use the Restriction list to provide one or more conditions restricting the list of objects to be described. To see the full list of restrictions and to which metadata kinds they are applicable, you should assign the Restrictions value to the MetaDataKind property and view the result.

See Also

- MetaDataKind

Request Support DAC Forum Provide Feedback

5.10.1.16.3 Methods

Methods of the **TDAMetaData** class.

For a complete list of the **TDAMetaData** class members, see the TDAMetaData Members topic.

Public

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.

EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds	Used to get values acceptable in the MetaDataKind property.
GetRestrictions	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the

	dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TDAMetaData Class
- TDAMetaData Class Members

Request Support DAC Forum Provide Feedback

5.10.1.16.3.1 GetMetaDataKinds Method

Used to get values acceptable in the MetaDataKind property.

Class

TDAMetaData

Syntax

```
procedure GetMetaDataKinds(List: TStrings);
```

Parameters

List

Holds the object that will be filled with metadata kinds (restrictions).

Remarks

Call the GetMetaDataKinds method to get values acceptable in the MetaDataKind property. The List parameter will be cleared and then filled with values.

See Also

- MetaDataKind

Request Support DAC Forum Provide Feedback

5.10.1.16.3.2 GetRestrictions Method

Used to find out which restrictions are applicable to a certain MetaDataKind.

Class

TDAMetaData

Syntax

```
procedure GetRestrictions(List: TStrings; const MetaDataKind:
string);
```

Parameters

List

Holds the object that will be filled with metadata kinds (restrictions).

MetaDataKind

Holds the metadata kind for which restrictions are returned.

Remarks

Call the GetRestrictions method to find out which restrictions are applicable to a certain MetaDataKind. The List parameter will be cleared and then filled with values.

See Also

- Restrictions
- GetMetaDataKinds

Request Support DAC Forum Provide Feedback

5.10.1.17 TDAParam Class

A class that forms objects to represent the values of the parameters set.

For a list of all members of this type, see TDAParam members.

Unit

DBAccess

Syntax

```
TDAParam = class(TParam);
```

Remarks

Use the properties of TDAParam to set the value of a parameter. Objects that use parameters create TDAParam objects to represent these parameters. For example, TDAParam objects are used by TCustomDASQL, TCustomDADataSet.

TDAParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding and the way the field is displayed, edited, or calculated, that are not needed in a TDAParam object. Conversely, TDAParam includes properties that indicate how the field value is passed as a parameter.

See Also

- TCustomDADataSet
- TCustomDASQL
- TDAParams

5.10.1.17.1 Members

TDAParam class overview.

Properties

Name	Description

AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.
AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
DataType	Indicates the data type of the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.
ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.
Value	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField	Assigns field name and field value to a param.
AssignFieldValue	Assigns the specified field properties and value to a parameter.
LoadFromFile	Places the content of a specified file into a TDAParam object.
LoadFromStream	Places the content from a stream into a TDAParam object.
SetBlobData	Overloaded. Writes the data from a specified buffer to BLOB.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2 Properties

Properties of the **TDAParam** class.

For a complete list of the **TDAParam** class members, see the TDAParam Members topic.

Public

Name	Description
AsBlob	Used to set and read the value of the BLOB parameter as string.
AsBlobRef	Used to set and read the value of the BLOB parameter as a TBlob object.
AsFloat	Used to assign the value for a float field to a parameter.
AsInteger	Used to assign the value for an integer field to the parameter.
AsLargeInt	Used to assign the value for a LargeInteger field to the parameter.

AsMemo	Used to assign the value for a memo field to the parameter.
AsMemoRef	Used to set and read the value of the memo parameter as a TBlob object.
AsSQLTimeStamp	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString	Used to assign the string value to the parameter.
AsWideString	Used to assign the Unicode string value to the parameter.
IsNull	Used to indicate whether the value assigned to a parameter is NULL.

Published

Name	Description
DataType	Indicates the data type of the parameter.
ParamType	Used to indicate the type of use for a parameter.
Size	Specifies the size of a string type parameter.
Value	Used to represent the value of the parameter as Variant.

See Also

- TDAParam Class
- TDAParam Class Members

5.10.1.17.2.1 AsBlob Property

Used to set and read the value of the BLOB parameter as string.

Class

TDAParam

Syntax

```
property AsBlob: TBlobData;
```

Remarks

Use the AsBlob property to set and read the value of the BLOB parameter as string. Setting AsBlob will set the DataType property to ftBlob. AsBlob is the value of the parameter when it represents the value of the LONG RAW type.

Request Support DAC Forum Provide Feedback

5.10.1.17.2.2 AsBlobRef Property

Used to set and read the value of the BLOB parameter as a TBlob object.

Class

TDAParam

Syntax

```
property AsBlobRef: TBlob;
```

Remarks

Use the AsBlobRef property to set and read the value of the BLOB parameter as a TBlob object. Setting AsBlobRef will set the DataType property to ftBlob. Specifies the value of the parameter when it represents the value of the LONG RAW type.

Request Support DAC Forum Provide Feedback

5.10.1.17.2.3 AsFloat Property

Used to assign the value for a float field to a parameter.

Class

TDAParam

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to assign the value for a float field to the parameter. Setting AsFloat will set the DataType property to dtFloat.

Read the AsFloat property to determine the value that was assigned to an output parameter, represented as Double. The value of the parameter will be converted to the Double value if possible.

5.10.1.17.2.4 AsInteger Property

Used to assign the value for an integer field to the parameter.

Class

TDAParam

Syntax

```
property AsInteger: LongInt;
```

Remarks

Use the AsInteger property to assign the value for an integer field to the parameter. Setting AsInteger will set the DataType property to dtInteger.

Read the AsInteger property to determine the value that was assigned to an output parameter, represented as a 32-bit integer. The value of the parameter will be converted to the Integer value if possible.

5.10.1.17.2.5 AsLargeInt Property

Used to assign the value for a LargeInteger field to the parameter.

Class

TDAParam

Syntax

```
property AsLargeInt: Int64;
```

Remarks

Set the AsLargeInt property to assign the value for an Int64 field to the parameter. Setting AsLargeInt will set the DataType property to dtLargeint.

Read the AsLargeInt property to determine the value that was assigned to an output parameter, represented as a 64-bit integer. The value of the parameter will be converted to the Int64 value if possible.

5.10.1.17.2.6 AsMemo Property

Used to assign the value for a memo field to the parameter.

Class

TDAParam

Syntax

```
property AsMemo: string;
```

Remarks

Use the AsMemo property to assign the value for a memo field to the parameter. Setting AsMemo will set the DataType property to ftMemo. AsMemo is the value of the parameter

when it represents the value of the LONG type.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.7 AsMemoRef Property

Used to set and read the value of the memo parameter as a TBlob object.

Class

TDAParam

Syntax

```
property AsMemoRef: TBlob;
```

Remarks

Use the AsMemoRef property to set and read the value of the memo parameter as a TBlob object. Setting AsMemoRef will set the DataType property to ftMemo. Specifies the value of the parameter when it represents the value of the LONG type.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.8 AsSQLTimeStamp Property

Used to specify the value of the parameter when it represents a SQL timestamp field.

Class

TDAParam

Syntax

```
property AsSQLTimeStamp: TSQLTimeStamp;
```

Remarks

Set the AsSQLTimeStamp property to assign the value for a SQL timestamp field to the parameter. Setting AsSQLTimeStamp sets the DataType property to ftTimeStamp.

 Request Support DAC Forum Provide Feedback

Reserved.

5.10.1.17.2.9 AsString Property

Used to assign the string value to the parameter.

Class

TDAParam

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to the parameter. Setting AsString will set the DataType property to ftString.

Read the AsString property to determine the value that was assigned to an output parameter represented as a string. The value of the parameter will be converted to a string.

Request Support DAC Forum Provide Feedback

5.10.1.17.2.10 AsWideString Property

Used to assign the Unicode string value to the parameter.

Class

TDAParam

Syntax

```
property AsWideString: string;
```

Remarks

Set AsWideString to assign the Unicode string value to the parameter. Setting AsWideString will set the DataType property to ftWideString.

Read the AsWideString property to determine the value that was assigned to an output parameter, represented as a Unicode string. The value of the parameter will be converted to a

Unicode string.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.11 DataType Property

Indicates the data type of the parameter.

Class

TDAParam

Syntax

```
property DataType: TFieldType stored IsDataTypeStored;
```

Remarks

DataType is set automatically when a value is assigned to a parameter. Do not set DataType for bound fields, as this may cause the assigned value to be misinterpreted.

Read DataType to learn the type of data that was assigned to the parameter. Every possible value of DataType corresponds to the type of a database field.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.12 IsNull Property

Used to indicate whether the value assigned to a parameter is NULL.

Class

TDAParam

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to indicate whether the value assigned to a parameter is NULL.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.13 ParamType Property

Used to indicate the type of use for a parameter.

Class

TDAParam

Syntax

```
property ParamType default DB . ptUnknown;
```

Remarks

Objects that use TDAParam objects to represent field parameters set ParamType to indicate the type of use for a parameter.

To learn the description of TParamType refer to Delphi Help.

Note: The value of ParamType is important for the LONG, LONG RAW, BLOB and CLOB parameters. To write data to database, set ptInput to ParamType, to read data from database, set ptOutput to ParamType.

Request Support DAC Forum Provide Feedback

5.10.1.17.2.14 Size Property

Specifies the size of a string type parameter.

Class

TDAParam

Syntax

```
property Size: integer default 0;
```

Remarks

Use the Size property to indicate the maximum number of characters the parameter may

contain. Use the Size property only for Output parameters of the **ftString, ftFixedChar, ftBytes, ftVarBytes,** or **ftWideString** type.

 Request Support DAC Forum Provide Feedback

5.10.1.17.2.15 Value Property

Used to represent the value of the parameter as Variant.

Class

TDAParam

Syntax

```
property Value: variant stored IsValueStored;
```

Remarks

The Value property represents the value of the parameter as Variant.

Use Value in generic code that manipulates the values of parameters without the need to know the field type the parameter represent.

 Request Support DAC Forum Provide Feedback

5.10.1.17.3 Methods

Methods of the **TDAParam** class.

For a complete list of the **TDAParam** class members, see the TDAParam Members topic.

Public

Name	Description
AssignField	Assigns field name and field value to a param.
AssignFieldValue	Assigns the specified field properties and value to a parameter.
LoadFromFile	Places the content of a

	specified file into a TDAParam object.
LoadFromStream	Places the content from a stream into a TDAParam object.
SetBlobData	Overloaded. Writes the data from a specified buffer to BLOB.

See Also

- TDAParam Class
- TDAParam Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.17.3.1 AssignField Method

Assigns field name and field value to a param.

Class

TDAParam

Syntax

```
procedure AssignField(Field: TField);
```

Parameters

Field

Holds the field which name and value should be assigned to the param.

Remarks

Call the AssignField method to assign field name and field value to a param.

 Request Support DAC Forum Provide Feedback

5.10.1.17.3.2 AssignFieldValue Method

Assigns the specified field properties and value to a parameter.

Class

TDAParam

Syntax

```
procedure AssignFieldValue(Field: TField; const Value: Variant);
virtual;
```

Parameters

Field

Holds the field the properties of which will be assigned to the parameter.

Value

Holds the value for the parameter.

Remarks

Call the AssignFieldValue method to assign the specified field properties and value to a parameter.

5.10.1.17.3.3 LoadFromFile Method

Places the content of a specified file into a TDAParam object.

Class

TDAParam

Syntax

```
procedure LoadFromFile(const FileName: string; BlobType:
TBlobType);
```

Parameters

FileName

Holds the name of the file.

BlobType

Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

Remarks

Use the LoadFromFile method to place the content of a file specified by FileName into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

See Also

- LoadFromStream

Request Support DAC Forum Provide Feedback

5.10.1.17.3.4 LoadFromStream Method

Places the content from a stream into a TDAParam object.

Class

TDAParam

Syntax

```
procedure LoadFromStream(Stream: TStream; BlobType: TBlobType);
virtual;
```

Parameters

Stream

Holds the stream to copy content from.

BlobType

Holds a value that modifies the DataType property so that this TDAParam object now holds the BLOB value.

Remarks

Call the LoadFromStream method to place the content from a stream into a TDAParam object. The BlobType value modifies the DataType property so that this TDAParam object now holds the BLOB value.

See Also

- LoadFromFile

5.10.1.17.3.5 SetBlobData Method

Writes the data from a specified buffer to BLOB.

Class

TDAParam

Overload List

Name	Description
SetBlobData(Buffer: TValueBuffer)	Writes the data from a specified buffer to BLOB.
SetBlobData(Buffer: IntPtr; Size: Integer)	Writes the data from a specified buffer to BLOB.

Writes the data from a specified buffer to BLOB.

Class

TDAParam

Syntax

```
procedure SetBlobData(Buffer: TValueBuffer); overload;
```

Parameters

Buffer

Holds the pointer to the data.

Writes the data from a specified buffer to BLOB.

Class

TDAParam

Syntax

```
procedure SetBlobData(Buffer: IntPtr; Size: Integer); overload;
```

Parameters

Buffer

Holds the pointer to data.

Size

Holds the number of bytes to read from the buffer.

Remarks

Call the SetBlobData method to write data from a specified buffer to BLOB.

Request Support DAC Forum Provide Feedback

5.10.1.18 TDAParams Class

This class is used to manage a list of TDAParam objects for an object that uses field parameters.

For a list of all members of this type, see TDAParams members.

Unit

DBAccess

Syntax

```
TDAParams = class(TParams);
```

Remarks

Use TDAParams to manage a list of TDAParam objects for an object that uses field parameters. For example, TCustomDADataSet objects and TCustomDASQL objects use TDAParams objects to create and access their parameters.

See Also

- TCustomDADataSet.Params
- TCustomDASQL.Params
- TDAParam

 Request Support DAC Forum Provide Feedback

5.10.1.18.1 Members

TDAParams class overview.

Properties

Name	Description
Items	Used to interate through all parameters.

Methods

Name	Description
FindParam	Searches for a parameter with the specified name.
ParamByName	Searches for a parameter with the specified name.

 Request Support DAC Forum Provide Feedback

5.10.1.18.2 Properties

Properties of the **TDAParams** class.

For a complete list of the **TDAParams** class members, see the TDAParams Members topic.

Public

Name	Description
Items	Used to interate through all parameters.

See Also

- TDAParams Class
- TDAParams Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.18.2.1 Items Property(Indexer)

Used to interate through all parameters.

Class

TDAParams

Syntax

```
property Items[Index: integer]: TDAParam; default;
```

Parameters

Index

Holds an index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all parameters. Index identifies the index in the range 0..Count - 1. Items can reference a particular parameter by its index, but the ParamByName method is preferred in order to avoid depending on the order of the parameters.

 Request Support DAC Forum Provide Feedback

5.10.1.18.3 Methods

Methods of the **TDAParams** class.

For a complete list of the **TDAParams** class members, see the TDAParams Members topic.

Public

Name	Description

FindParam	Searches for a parameter with the specified name.
ParamByName	Searches for a parameter with the specified name.

See Also

- TDAParams Class
- TDAParams Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.18.3.1 FindParam Method

Searches for a parameter with the specified name.

Class

TDAParams

Syntax

```
function FindParam(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if a match was found. Nil otherwise.

Remarks

Use the FindParam method to find a parameter with the name passed in Value. If a match is found, FindParam returns the parameter. Otherwise, it returns nil. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate more than one parameter at a time by name, use the GetParamList method instead. To get only the value of a named parameter, use the ParamValues property.

 Request Support DAC Forum Provide Feedback

5.10.1.18.3.2 ParamByName Method

Searches for a parameter with the specified name.

Class

TDAParams

Syntax

```
function ParamByName(const Value: string): TDAParam;
```

Parameters

Value

Holds the parameter name.

Return Value

a parameter, if the match was found. otherwise an exception is raised.

Remarks

Use the ParamByName method to find a parameter with the name passed in Value. If a match was found, ParamByName returns the parameter. Otherwise, an exception is raised. Use this method rather than a direct reference to the Items property to avoid depending on the order of the entries.

To locate a parameter by name without raising an exception if the parameter is not found, use the FindParam method.

5.10.1.19 TDATransaction Class

A base class that implements functionality for controlling transactions.

For a list of all members of this type, see TDATransaction members.

Unit

DBAccess

Syntax

```
TDATransaction = class(TComponent);
```

Remarks

TDATransaction is a base class for components implementing functionality for managing transactions.

Do not create instances of TDATransaction. Use descendants of the TDATransaction class instead.

Request Support DAC Forum Provide Feedback

5.10.1.19.1 Members

TDATransaction class overview.

Properties

Name	Description
Active	Used to determine if the transaction is active.
DefaultCloseAction	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Methods

Name	Description
Commit	Commits the current transaction.
Rollback	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction	Begins a new transaction.

Events

Name	Description

OnCommit	Occurs after the transaction has been successfully committed.
OnCommitRetaining	Occurs after CommitRetaining has been executed.
OnError	Used to process errors that occur during executing a transaction.
OnRollback	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining	Occurs after RollbackRetaining has been executed.

Request Support DAC Forum Provide Feedback

5.10.1.19.2 Properties

Properties of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the TDATransaction Members topic.

Public

Name	Description
Active	Used to determine if the transaction is active.
DefaultCloseAction	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

See Also

- TDATransaction Class
- TDATransaction Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.19.2.1 Active Property

Used to determine if the transaction is active.

Class

TDATransaction

Syntax

```
property Active: boolean;
```

Remarks

Indicates whether the transaction is active. This property is read-only.

 Request Support DAC Forum Provide Feedback

5.10.1.19.2.2 DefaultCloseAction Property

Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

Class

TDATransaction

Syntax

```
property DefaultCloseAction: TCRTransactionAction default
taRollback;
```

Remarks

Use DefaultCloseAction to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.

 Request Support DAC Forum Provide Feedback

5.10.1.19.3 Methods

Methods of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the TDATransaction Members topic.

Public

Name	Description
Commit	Commits the current transaction.
Rollback	Discards all modifications of data associated with the current transaction and ends the transaction.
StartTransaction	Begins a new transaction.

See Also

- TDATransaction Class
- TDATransaction Class Members

Request Support DAC Forum Provide Feedback

5.10.1.19.3.1 Commit Method

Commits the current transaction.

Class

TDATransaction

Syntax

```
procedure Commit; virtual;
```

Remarks

Call the Commit method to commit the current transaction. On commit server writes permanently all pending data updates associated with the current transaction to the database,

and then finishes the transaction.

See Also

- Rollback
- StartTransaction

 Request Support DAC Forum Provide Feedback

5.10.1.19.3.2 Rollback Method

Discards all modifications of data associated with the current transaction and ends the transaction.

Class

TDATransaction

Syntax

```
procedure Rollback; virtual;
```

Remarks

Call Rollback to cancel all data modifications made within the current transaction to the database server, and finish the transaction.

See Also

- Commit
- StartTransaction

 Request Support DAC Forum Provide Feedback

5.10.1.19.3.3 StartTransaction Method

Begins a new transaction.

Class

TDATransaction

Syntax

```
procedure StartTransaction; virtual;
```

Remarks

Call the StartTransaction method to begin a new transaction against the database server. Before calling StartTransaction, an application should check the Active property. If TDATransaction.Active is True, indicating that a transaction is already in progress, a subsequent call to StartTransaction will raise EDatabaseError. An active transaction must be finished by call to Commit or Rollback before call to StartTransaction. Call to StartTransaction when connection is closed also will raise EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until the application calls Commit to save the changes, or Rollback to cancel them.

See Also

- Commit
- Rollback

5.10.1.19.4 Events

Events of the **TDATransaction** class.

For a complete list of the **TDATransaction** class members, see the TDATransaction Members topic.

Public

Name	Description
OnCommit	Occurs after the transaction has been successfully committed.
OnCommitRetaining	Occurs after CommitRetaining has been executed.
OnError	Used to process errors that

	occur during executing a transaction.
OnRollback	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining	Occurs after RollbackRetaining has been executed.

See Also

- TDATransaction Class
- TDATransaction Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.19.4.1 OnCommit Event

Occurs after the transaction has been successfully committed.

Class

TDATransaction

Syntax

```
property OnCommit: TNotifyEvent;
```

Remarks

The OnCommit event fires when the M:Devart.Dac.TDATransaction.Commit method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Odac.TOraTransaction.CommitRetaining() method execution, the OnCommitRetaining event is used. When an error occurs during commit, the OnError event fires.

See Also

- Commit
- OnError

 Request Support DAC Forum Provide Feedback

5.10.1.19.4.2 OnCommitRetaining Event

Occurs after CommitRetaining has been executed.

Class

TDATransaction

Syntax

```
property OnCommitRetaining: TNotifyEvent;
```

Remarks

The OnCommitRetaining event fires when the CommitRetaining method is executed, just after the transaction is successfully committed. In order to respond to the M:Devart.Dac.TDATransaction.Commit method execution, the OnCommit event is used.

When an error occurs during commit, the OnError event fired.

See Also

- Commit
- OnCommit
- OnError

 Request Support DAC Forum Provide Feedback

5.10.1.19.4.3 OnError Event

Used to process errors that occur during executing a transaction.

Class

TDATransaction

Syntax

```
property OnError: TDATransactionErrorEvent;
```

Remarks

Add a handler to the OnError event to process errors that occur during executing a transaction and save point control statements such as Commit, Rollback, TOraTransaction.Savepoint, TOraTransaction.RollbackToSavepoint, and others. Check the E parameter to get the error code.

See Also

- Commit
- Rollback
- StartTransaction

Request Support DAC Forum Provide Feedback

5.10.1.19.4.4 OnRollback Event

Occurs after the transaction has been successfully rolled back.

Class

TDATransaction

Syntax

```
property OnRollback: TNotifyEvent;
```

Remarks

The OnRollback event fires when the M:Devart.Dac.TDATransaction.Rollback method is executed, just after the transaction is successfully rolled back. In order to respond to the M:Devart.Odac.TOraTransaction.RollbackRetaining() method execution, the OnRollbackRetaining event is used.

When an error occurs during rollback, the OnError event fired.

See Also

- Rollback
- OnError

5.10.1.19.4.5 OnRollbackRetaining Event

Occurs after RollbackRetaining has been executed.

Class

TDATransaction

Syntax

```
property OnRollbackRetaining: TNotifyEvent;
```

Remarks

The OnRollbackRetaining event fires when the RollbackRetaining method is executed, just after the transaction is successfully rolled back. In order to respond to the Rollback method execution, the OnRollback event is used. When an error occurs during rollback, the OnError event fired.

See Also

- Rollback
- OnRollback
- OnError

5.10.1.20 TMacro Class

Object that represents the value of a macro.

For a list of all members of this type, see TMacro members.

Unit

DBAccess

Syntax

```
TMacro = class(TCollectionItem);
```

Remarks

TMacro object represents the value of a macro. Macro is a variable that holds string value. You just insert **&** MacroName in a SQL query text and change the value of macro by the Macro property editor at design time or the Value property at run time. At the time of opening query macro is replaced by its value.

If by any reason it is not convenient for you to use the ' **&** ' symbol as a character of macro replacement, change the value of the MacroChar variable.

See Also

- TMacros

Request Support DAC Forum Provide Feedback

5.10.1.20.1 Members

TMacro class overview.

Properties

Name	Description
Active	Used to determine if the macro should be expanded.
AsDateTime	Used to set the TDataTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

Request Support DAC Forum Provide Feedback

Reserved.

5.10.1.20.2 Properties

Properties of the **TMacro** class.

For a complete list of the **TMacro** class members, see the TMacro Members topic.

Public

Name	Description
AsDateTime	Used to set the TDataTime value to a macro.
AsFloat	Used to set the float value to a macro.
AsInteger	Used to set the integer value to a macro.
AsString	Used to assign the string value to a macro.

Published

Name	Description
Active	Used to determine if the macro should be expanded.
Name	Used to identify a particular macro.
Value	Used to set the value to a macro.

See Also

- TMacro Class
- TMacro Class Members

Request Support DAC Forum Provide Feedback

5.10.1.20.2.1 Active Property

Used to determine if the macro should be expanded.

Class

TMacro

Syntax

```
property Active: boolean default True;
```

Remarks

When set to True, the macro will be expanded, otherwise macro definition is replaced by null string. You can use the Active property to modify the SQL property.

The default value is True.

Example

```
OraQuery.SQL.Text := 'SELECT * FROM Dept WHERE DeptNo > 20 &Cond1';
OraQuery.Macros[0].Value := 'and DName is NULL';
OraQuery.Macros[0].Active:= False;
```


Request Support DAC Forum Provide Feedback

5.10.1.20.2.2 AsDateTime Property

Used to set the TDataTime value to a macro.

Class

TMacro

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDataTime property to set the TDataTime value to a macro.

Request Support DAC Forum Provide Feedback

5.10.1.20.2.3 AsFloat Property

Used to set the float value to a macro.

Class

TMacro

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to set the float value to a macro.

Request Support DAC Forum Provide Feedback

5.10.1.20.2.4 AsInteger Property

Used to set the integer value to a macro.

Class

TMacro

Syntax

```
property AsInteger: integer;
```

Remarks

Use the AsInteger property to set the integer value to a macro.

Request Support DAC Forum Provide Feedback

5.10.1.20.2.5 AsString Property

Used to assign the string value to a macro.

Class

TMacro

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to assign the string value to a macro. Read the AsString property to determine the value of macro represented as a string.

 Request Support DAC Forum Provide Feedback

5.10.1.20.2.6 Name Property

Used to identify a particular macro.

Class

TMacro

Syntax

```
property Name: string;
```

Remarks

Use the Name property to identify a particular macro.

 Request Support DAC Forum Provide Feedback

5.10.1.20.2.7 Value Property

Used to set the value to a macro.

Class

TMacro

Syntax

```
property Value: string;
```

Remarks

Use the Value property to set the value to a macro.

Request Support DAC Forum Provide Feedback

5.10.1.21 TMacros Class

Controls a list of TMacro objects for the TCustomDASQL.Macros or TCustomDADataSet components.

For a list of all members of this type, see TMacros members.

Unit

DBAccess

Syntax

```
TMacros = class(TCollection);
```

Remarks

Use TMacros to manage a list of TMacro objects for the TCustomDASQL or TCustomDADataSet components.

See Also

- TMacro

Request Support DAC Forum Provide Feedback

5.10.1.21.1 Members

TMacros class overview.

Properties

Name	Description
Items	Used to iterate through all the macros parameters.

Methods

Name	Description
AssignValues	Copies the macros values and properties from the specified source.
Expand	Changes the macros in the passed SQL statement to their values.
FindMacro	Finds a macro with the specified name.
IsEqual	Compares itself with another TMacro object.
MacroByName	Used to search for a macro with the specified name.
Scan	Creates a macros from the passed SQL statement.

 Request Support DAC Forum Provide Feedback

5.10.1.21.2 Properties

Properties of the **TMacros** class.

For a complete list of the **TMacros** class members, see the TMacros Members topic.

Public

Name	Description
Items	Used to iterate through all the macros parameters.

See Also

- TMacros Class
- TMacros Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.21.2.1 Items Property(Indexer)

Used to iterate through all the macros parameters.

Class

TMacros

Syntax

```
property Items[Index: integer]: TMacro; default;
```

Parameters

Index

Holds the index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all macros parameters. Index identifies the index in the range 0..Count - 1.

5.10.1.21.3 Methods

Methods of the **TMacros** class.

For a complete list of the **TMacros** class members, see the TMacros Members topic.

Public

Name	Description
AssignValues	Copies the macros values and properties from the specified source.
Expand	Changes the macros in the passed SQL statement to their values.
FindMacro	Finds a macro with the specified name.
IsEqual	Compares itself with another TMacro object.
MacroByName	Used to search for a macro

	with the specified name.
Scan	Creates a macros from the passed SQL statement.

See Also

- TMacros Class
- TMacros Class Members

 Request Support DAC Forum Provide Feedback

5.10.1.21.3.1 AssignValues Method

Copies the macros values and properties from the specified source.

Class

TMacros

Syntax

```
procedure AssignValues(Value: TMacros);
```

Parameters

Value

Holds the source to copy the macros values and properties from.

Remarks

The Assign method copies the macros values and properties from the specified source. Macros are not recreated. Only the values of macros with matching names are assigned.

 Request Support DAC Forum Provide Feedback

5.10.1.21.3.2 Expand Method

Changes the macros in the passed SQL statement to their values.

Class

TMacros

Syntax

```
procedure Expand(var SQL: string);
```

Parameters

SQL

Holds the passed SQL statement.

Remarks

Call the Expand method to change the macros in the passed SQL statement to their values.

 Request Support DAC Forum Provide Feedback

5.10.1.21.3.3 FindMacro Method

Finds a macro with the specified name.

Class

TMacros

Syntax

```
function FindMacro(const Value: string): TMacro;
```

Parameters

Value

Holds the value of a macro to search for.

Return Value

TMacro object if a match is found, nil otherwise.

Remarks

Call the FindMacro method to find a macro with the specified name. If a match is found, FindMacro returns the macro. Otherwise, it returns nil. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

 Request Support DAC Forum Provide Feedback

5.10.1.21.3.4 IsEqual Method

Compares itself with another TMacro object.

Class

TMacros

Syntax

```
function IsEqual(Value: TMacros): boolean;
```

Parameters

Value

Holds the values of TMacro objects.

Return Value

True, if the number of TMacro objects and the values of all TMacro objects are equal.

Remarks

Call the IsEqual method to compare itself with another TMacro object. Returns True if the number of TMacro objects and the values of all TMacro objects are equal.

Request Support DAC Forum Provide Feedback

5.10.1.21.3.5 MacroByName Method

Used to search for a macro with the specified name.

Class

TMacros

Syntax

```
function MacroByName(const Value: string): TMacro;
```

Parameters

Value

Holds a name of the macro to search for.

Return Value

TMacro object, if a macro with specified name was found.

Remarks

Call the MacroByName method to find a Macro with the name passed in Value. If a match is found, MacroByName returns the Macro. Otherwise, an exception is raised. Use this method instead of a direct reference to the Items property to avoid depending on the order of the items.

To locate a macro by name without raising an exception if the parameter is not found, use the FindMacro method.

To set a value to a macro, use the TMacro.Value property.

Request Support DAC Forum Provide Feedback

5.10.1.21.3.6 Scan Method

Creates a macros from the passed SQL statement.

Class

TMacros

Syntax

```
procedure Scan(const SQL: string);
```

Parameters

SQL

Holds the passed SQL statement.

Remarks

Call the Scan method to create a macros from the passed SQL statement. On that all existing TMacro objects are cleared.

Request Support DAC Forum Provide Feedback

5.10.1.22 TPoolingOptions Class

This class allows setting up the behaviour of the connection pool.

For a list of all members of this type, see TPoolingOptions members.

Unit

DBAccess

Syntax

```
TPoolingOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.10.1.22.1 Members

TPoolingOptions class overview.

Properties

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

 Request Support DAC Forum Provide Feedback

5.10.1.22.2 Properties

Properties of the **TPoolingOptions** class.

For a complete list of the **TPoolingOptions** class members, see the TPoolingOptions Members topic.

Published

Name	Description
ConnectionLifetime	Used to specify the maximum time during which an open connection can be used by connection pool.
MaxPoolSize	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId	Used to specify an ID for a connection pool.
Validate	Used for a connection to be validated when it is returned from the pool.

See Also

- TPoolingOptions Class
- TPoolingOptions Class Members

5.10.1.22.2.1 ConnectionLifetime Property

Used to specify the maximum time during which an open connection can be used by connection pool.

Class

TPoolingOptions

Syntax

```
property ConnectionLifetime: integer default
DefValConnectionLifetime;
```

Remarks

Use the ConnectionLifeTime property to specify the maximum time during which an open connection can be used by connection pool. Measured in milliseconds. Pool deletes connections with exceeded connection lifetime when TCustomDAConnection is about to close. If ConnectionLifetime is set to 0 (by default), then the lifetime of connection is infinite. ConnectionLifetime concerns only inactive connections in the pool.

Request Support DAC Forum Provide Feedback

5.10.1.22.2.2 MaxPoolSize Property

Used to specify the maximum number of connections that can be opened in connection pool.

Class

TPoolingOptions

Syntax

```
property MaxPoolSize: integer default DefValMaxPoolSize;
```

Remarks

Specifies the maximum number of connections that can be opened in connection pool. Once this value is reached, no more connections are opened. The valid values are 1 and higher.

Request Support DAC Forum Provide Feedback

5.10.1.22.2.3 MinPoolSize Property

Used to specify the minimum number of connections that can be opened in the connection pool.

Class

TPoolingOptions

Syntax

```
property MinPoolSize: integer default DefValMinPoolSize;
```

Remarks

Use the MinPoolSize property to specify the minimum number of connections that can be opened in the connection pool.

Request Support DAC Forum Provide Feedback

5.10.1.22.2.4 PoolId Property

Used to specify an ID for a connection pool.

Class

TPoolingOptions

Syntax

```
property PoolId: Integer default DefValPoolId;
```

Remarks

Use the PoolId property to make a group of connections to use a specific connection pool.

Request Support DAC Forum Provide Feedback

5.10.1.22.2.5 Validate Property

Used for a connection to be validated when it is returned from the pool.

Class

TPoolingOptions

Syntax

```
property Validate: boolean default DefValValidate;
```

Remarks

If the Validate property is set to True, connection will be validated when it is returned from the pool. By default this option is set to False and pool does not validate connection when it is returned to be used by a TCustomDAConnection component.

 Request Support DAC Forum Provide Feedback

5.10.1.23 TSmartFetchOptions Class

Smart fetch options are used to set up the behavior of the SmartFetch mode.

For a list of all members of this type, see TSmartFetchOptions members.

Unit

DBAccess

Syntax

```
TSmartFetchOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.10.1.23.1 Members

TSmartFetchOptions class overview.

Properties

Name	Description
Enabled	Sets SmartFetch mode enabled or not.
LiveBlock	Used to minimize memory consumption.
PrefetchedFields	List of fields additional to key fields that will be read from the database on dataset open.

SQLGetKeyValues	SQL query for the read key and prefetched fields from the database.

Request Support DAC Forum Provide Feedback

5.10.1.23.2 Properties

Properties of the **TSmartFetchOptions** class.

For a complete list of the **TSmartFetchOptions** class members, see the TSmartFetchOptions Members topic.

Published

Name	Description
Enabled	Sets SmartFetch mode enabled or not.
LiveBlock	Used to minimize memory consumption.
PrefetchedFields	List of fields additional to key fields that will be read from the database on dataset open.
SQLGetKeyValues	SQL query for the read key and prefetched fields from the database.

See Also

- TSmartFetchOptions Class
- TSmartFetchOptions Class Members

Request Support DAC Forum Provide Feedback

5.10.1.23.2.1 Enabled Property

Sets SmartFetch mode enabled or not.

Class

TSmartFetchOptions

Syntax

```
property Enabled: Boolean default False;
```

 Request Support DAC Forum Provide Feedback

5.10.1.23.2.2 LiveBlock Property

Used to minimize memory consumption.

Class

TSmartFetchOptions

Syntax

```
property LiveBlock: Boolean default True;
```

Remarks

If LiveBlock is True, then on navigating through a dataset forward or backward, memory will be allocated for records count defined in the the FetchRows propety, and no additional memory will be allocated. But if you return records that were read from the database before, they will be read from the database again, because when you left block with these records, memory was free. So the LiveBlock mode minimizes memory consumption, but can decrease performance, because it can lead to repeated data reading from the database.

The default value of LiveBlock is False.

 Request Support DAC Forum Provide Feedback

5.10.1.23.2.3 PrefetchedFields Property

List of fields additional to key fields that will be read from the database on dataset open.

Class

TSmartFetchOptions

Syntax

```
property PrefetchedFields: string;
```

Remarks

If you are going to use locate, filter or sort by some fields, then these fields should be added to the prefetched fields list to avoid excessive reading from the database.

Request Support DAC Forum Provide Feedback

5.10.1.23.2.4 SQLGetKeyValues Property

SQL query for the read key and prefetched fields from the database.

Class

TSmartFetchOptions

Syntax

```
property SQLGetKeyValues: TStrings;
```

Remarks

SQLGetKeyValues is used when the basic SQL query is complex and the query for reading the key and prefetched fields can't be generated automatically.

Request Support DAC Forum Provide Feedback

5.10.2 Types

Types in the **DBAccess** unit.

Types

Name	Description
TAfterExecuteEvent	This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecu

	te events.
TAfterFetchEvent	This type is used for the TCustomDADataSet.AfterFetch event.
TBeforeFetchEvent	This type is used for the TCustomDADataSet.BeforeFetch event.
TConnectionLostEvent	This type is used for the TCustomDAConnection.OnConnectionLost event.
TDAConnectionErrorEvent	This type is used for the TCustomDAConnection.OnError event.
TDATransactionErrorEvent	This type is used for the TDATransaction.OnError event.
TRefreshOptions	Represents the set of TRefreshOption.
TUpdateExecuteEvent	This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

 Request Support DAC Forum Provide Feedback

5.10.2.1 TAfterExecuteEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterExecute and TCustomDASQL.AfterExecute events.

Unit

DBAccess

Syntax

```
TAfterExecuteEvent = procedure (Sender: TObject; Result: boolean)
of object;
```

Parameters

Sender

An object that raised the event.

Result

The result is True if SQL statement is executed successfully. False otherwise.

Request Support DAC Forum Provide Feedback

5.10.2.2 TAfterFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterFetch event.

Unit

DBAccess

Syntax

```
TAfterFetchEvent = procedure (DataSet: TCustomDADataSet) of object;
```

Parameters

DataSet

Holds the TCustomDADataSet descendant to synchronize the record position with.

Request Support DAC Forum Provide Feedback

5.10.2.3 TBeforeFetchEvent Procedure Reference

This type is used for the TCustomDADataSet.BeforeFetch event.

Unit

DBAccess

Syntax

```
TBeforeFetchEvent = procedure (DataSet: TCustomDADataSet; var Cancel: boolean) of object;
```

Parameters

DataSet

Holds the TCustomDADataSet descendant to synchronize the record position with.

Cancel

True, if the current fetch operation should be aborted.

 Request Support DAC Forum Provide Feedback

5.10.2.4 TConnectionLostEvent Procedure Reference

This type is used for the TCustomDAConnection.OnConnectionLost event.

Unit

DBAccess

Syntax

```
TConnectionLostEvent = procedure (Sender: TObject; Component:
TComponent; ConnLostCause: TConnLostCause; var RetryMode:
TRetryMode) of object;
```

Parameters

Sender

An object that raised the event.

Component

ConnLostCause

The reason of the connection loss.

RetryMode

The application behavior when connection is lost.

 Request Support DAC Forum Provide Feedback

5.10.2.5 TDAConnectionErrorEvent Procedure Reference

This type is used for the TCustomDAConnection.OnError event.

Unit

DBAccess

Syntax

```
TDAConnectionErrorEvent = procedure (Sender: TObject; E: EDAError;
var Fail: boolean) of object;
```

Parameters

Sender
An object that raised the event.

E
The error information.

Fail
False, if an error dialog should be prevented from being displayed and EAbort exception should be raised to cancel current operation .

 Request Support DAC Forum Provide Feedback

5.10.2.6 TDATransactionErrorEvent Procedure Reference

This type is used for the TDATransaction.OnError event.

Unit

DBAccess

Syntax

```
TDATransactionErrorEvent = procedure (Sender: TObject; E:
EDAError; var Fail: boolean) of object;
```

Parameters

Sender
An object that raised the event.

E
The error code.

Fail
False, if an error dialog should be prevented from being displayed and EAbort exception to cancel the current operation should be raised.

 Request Support DAC Forum Provide Feedback

5.10.2.7 TRefreshOptions Set

Represents the set of TRefreshOption.

Unit

DBAccess

Syntax

```
TRefreshOptions = set of TRefreshOption;
```

 Request Support DAC Forum Provide Feedback

5.10.2.8 TUpdateExecuteEvent Procedure Reference

This type is used for the TCustomDADataSet.AfterUpdateExecute and TCustomDADataSet.BeforeUpdateExecute events.

Unit

DBAccess

Syntax

```
TUpdateExecuteEvent = procedure (Sender: TDataSet; StatementTypes:
TStatementTypes; Params: TDAParams) of object;
```

Parameters

Sender
An object that raised the event.

StatementTypes
Holds the type of the SQL statement being executed.

Params
Holds the parameters with which the SQL statement will be executed.

 Request Support DAC Forum Provide Feedback

5.10.3 Enumerations

Enumerations in the **DBAccess** unit.

Enumerations

Name	Description
TCheckMode	Specifies the action to take when another user makes modifications to a record.

TLabelSet	Sets the languauge of labels in the connect dialog.
TLockMode	Specifies the lock mode.
TRefreshOption	Indicates when the editing record will be refreshed.
TRetryMode	Specifies the application behavior when connection is lost.

Request Support DAC Forum Provide Feedback

5.10.3.1 TCheckMode Enumeration

Specifies the action to take when another user makes modifications to a record.

Unit

DBAccess

Syntax

```
TCheckMode = (cmNone, cmException, cmRefresh);
```

Values

Value	Meaning
cmException	If a record was changed, TOraDataSet raises an exception.
cmNone	No check is performed. The default value.
cmRefresh	If a record was changed, TOraDataSet refreshes it.

Request Support DAC Forum Provide Feedback

5.10.3.2 TLabelSet Enumeration

Sets the languauge of labels in the connect dialog.

Unit

DBAccess

Syntax

```
TLabelSet = (lsCustom, lsEnglish, lsFrench, lsGerman, lsItalian,
lsPolish, lsPortuguese, lsRussian, lsSpanish);
```

Values

Value	Meaning
lsCustom	Set the language of labels in the connect dialog manually.
lsEnglish	Set English as the language of labels in the connect dialog.
lsFrench	Set French as the language of labels in the connect dialog.
lsGerman	Set German as the language of labels in the connect dialog.
lsItalian	Set Italian as the language of labels in the connect dialog.
lsPolish	Set Polish as the language of labels in the connect dialog.
lsPortuguese	Set Portuguese as the language of labels in the connect dialog.
lsRussian	Set Russian as the language of labels in the connect dialog.
lsSpanish	Set Spanish as the language of labels in the connect dialog.

Request Support DAC Forum Provide Feedback

5.10.3.3 TLockMode Enumeration

Specifies the lock mode.

Unit

DBAccess

Syntax

```
TLockMode = (lmNone);
```

Values

Value	Meaning
lmLockDelayed	Locking occurs when the user posts an edited record, then the lock is released. Locking is done by the RefreshRecord method. Corresponds to optimistic locking.
lmLockImmediate	Locking occurs when the user starts editing a record. The lock is released after the user has posted or canceled the changes. Corresponds to pessimistic locking.

lmNone	No locking occurs. This mode should only be used in single user applications. The default value.

Request Support DAC Forum Provide Feedback

5.10.3.4 TRefreshOption Enumeration

Indicates when the editing record will be refreshed.

Unit

DBAccess

Syntax

```
TRefreshOption = (roAfterInsert, roAfterUpdate, roBeforeEdit);
```

Values

Value	Meaning
roAfterInsert	Refresh is performed after inserting.
roAfterUpdate	Refresh is performed after updating.
roBeforeEdit	Refresh is performed by Edit method.

Request Support DAC Forum Provide Feedback

5.10.3.5 TRetryMode Enumeration

Specifies the application behavior when connection is lost.

Unit

DBAccess

Syntax

```
TRetryMode = (rmRaise, rmReconnect, rmReconnectExecute);
```

Values

Value	Meaning

rmRaise	An exception is raised.
rmReconnect	Reconnect is performed and then exception is raised.
rmReconnectExecute	Reconnect is performed and abortive operation is reexecuted. Exception is not raised.

5.10.4 Variables

Variables in the **DBAccess** unit.

Variables

Name	Description
BaseSQLOldBehavior	After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.
ChangeCursor	When set to True allows data access components to change screen cursor for the execution time.
SQLGeneratorCompatibility	The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

5.10.4.1 BaseSQLOldBehavior Variable

After assigning SQL text and modifying it by AddWhere, DeleteWhere, and SetOrderBy, all subsequent changes of the SQL property will not be reflected in the BaseSQL property.

Unit

DBAccess

Syntax

```
BaseSQLOldBehavior: boolean = False;
```

Remarks

The BaseSQL property is similar to the SQL property, but it does not store changes made by the AddWhere, DeleteWhere, and SetOrderBy methods. After assigning SQL text and modifying it by one of these methods, all subsequent changes of the SQL property will not be reflected in the BaseSQL property. This behavior was changed in ODAC 5.55.1.26. To restore old behavior, set the BaseSQLOldBehavior variable to True.

 Request Support DAC Forum Provide Feedback

5.10.4.2 ChangeCursor Variable

When set to True allows data access components to change screen cursor for the execution time.

Unit

DBAccess

Syntax

```
ChangeCursor: boolean = True;
```

 Request Support DAC Forum Provide Feedback

5.10.4.3 SQLGeneratorCompatibility Variable

The value of the TCustomDADataSet.BaseSQL property is used to complete the refresh SQL statement, if the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause.

Unit

DBAccess

Syntax

```
SQLGeneratorCompatibility: boolean = False;
```

Remarks

If the manually assigned TCustomDAUpdateSQL.RefreshSQL property contains only WHERE clause, ODAC uses the value of the TCustomDADataSet.BaseSQL property to complete the refresh SQL statement. In this situation all modifications applied to the SELECT query by functions TCustomDADataSet.AddWhere, TCustomDADataSet.DeleteWhere are not taken into account. This behavior was changed in ODAC 6.00.0.4. To restore the old behavior, set the BaseSQLOldBehavior variable to True.

5.11 MemData

This unit contains classes for storing data in memory.

Classes

Name	Description
TAttribute	Holds the description of object attributes.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.
TMemData	Implements in-memory database.
TObjectType	Holds description object type and its attributes.
TSharedObject	A base class that allows to

	simplify memory management for object referenced by several other objects.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption.
TUpdateRecKinds	Represents the set of TUpdateRecKind.

Enumerations

Name	Description
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.
TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx method.
TSortType	Specifies a sort type for string fields.
TUpdateRecKind	Indicates records for which the ApplyUpdates method will be performed.

5.11.1 Classes

Classes in the **MemData** unit.

Classes

Name	Description
TAttribute	Holds the description of object attributes.
TBlob	Holds large object value for field and parameter dtBlob, dtMemo data types.
TCompressedBlob	Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.
TDBObject	A base class for classes that work with user-defined data types that have attributes.
TMemData	Implements in-memory database.
TObjectType	Holds description object type and its attributes.
TSharedObject	A base class that allows to simplify memory management for object referenced by several other objects.

Request Support DAC Forum Provide Feedback

5.11.1.1 TAttribute Class

Holds the description of object attributes.

For a list of all members of this type, see TAttribute members.

Unit

MemData

Syntax

```
TAttribute = class(System.TObject);
```

Remarks

The TAttribute class holds the description of object attributes. You can use TObjectType.Attibutes to access individual attributes. To create TAttribute objects call the TOraType.Describe method. It is called implicitly when ODAC fetches Oracle objects.

See Also

- TObjectType
- TOraType

Request Support DAC Forum Provide Feedback

5.11.1.1.1 Members

TAttribute class overview.

Properties

Name	Description
AttributeNo	Returns an attribute's ordinal position in object.
DataSize	Returns the size of an attribute value in internal representation.
DataType	Returns the type of data that was assigned to the Attribute.
Length	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
ObjectType	Returns a TObjectType object for an object attribute.
Offset	Returns an offset of the attribute value in internal representation.
Owner	Indicates TObjectType that uses the attribute to represent one of its

	attributes.
Scale	Returns the scale of dtFloat and dtInteger attributes.
Size	Returns the size of an attribute value in external representation.

 Request Support DAC Forum Provide Feedback

5.11.1.1.2 Properties

Properties of the **TAttribute** class.

For a complete list of the **TAttribute** class members, see the TAttribute Members topic.

Public

Name	Description
AttributeNo	Returns an attribute's ordinal position in object.
DataSize	Returns the size of an attribute value in internal representation.
DataType	Returns the type of data that was assigned to the Attribute.
Length	Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.
ObjectType	Returns a TObjectType object for an object attribute.
Offset	Returns an offset of the attribute value in internal representation.
Owner	Indicates TObjectType that uses the attribute to represent one of its attributes.
Scale	Returns the scale of dtFloat and dtInteger attributes.
Size	Returns the size of an attribute value in external

	representation.

See Also

- TAttribute Class
- TAttribute Class Members

Request Support DAC Forum Provide Feedback

5.11.1.1.2.1 AttributeNo Property

Returns an attribute's ordinal position in object.

Class

TAttribute

Syntax

```
property AttributeNo: Word;
```

Remarks

Use the AttributeNo property to learn an attribute's ordinal position in object, where 1 is the first field.

See Also

- TObjectType.Attributes

Request Support DAC Forum Provide Feedback

5.11.1.1.2.2 DataSize Property

Returns the size of an attribute value in internal representation.

Class

TAttribute

Syntax

```
property DataSize: Integer;
```

Remarks

Use the DataSize property to learn the size of an attribute value in internal representation.

For example:

dtDate	17 (sizeof(OCIDate)
dtFloat	22 (sizeof(OCINumber))
dtInteger	22 (sizeof(OCINumber))

See Also

- TOraObject.Instance
- Offset

Request Support DAC Forum Provide Feedback

5.11.1.1.2.3 DataType Property

Returns the type of data that was assigned to the Attribute.

Class

TAttribute

Syntax

```
property DataType: Word;
```

Remarks

Use the DataType property to discover the type of data that was assigned to the Attribute.

Possible values: dtDate, dtFloat, dtInteger, dtString, dtObject.

Request Support DAC Forum Provide Feedback

5.11.1.1.2.4 Length Property

Returns the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

Class

TAttribute

Syntax

```
property Length: Word;
```

Remarks

Use the Length property to learn the length of the string for dtString attribute and precision for dtInteger and dtFloat attribute.

See Also

- Scale

 Request Support DAC Forum Provide Feedback

5.11.1.1.2.5 ObjectType Property

Returns a TObjectType object for an object attribute.

Class

TAttribute

Syntax

```
property ObjectType: TObjectType;
```

Remarks

Use the ObjectType property to return a TObjectType object for an object attribute.

 Request Support DAC Forum Provide Feedback

5.11.1.1.2.6 Offset Property

Returns an offset of the attribute value in internal representation.

Class

TAttribute

Syntax

```
property Offset: Integer;
```

Remarks

Use the DataSize property to learn an offset of the attribute value in internal representation.

See Also

- TOraObject.Instance
- DataSize

 Request Support DAC Forum Provide Feedback

5.11.1.1.2.7 Owner Property

Indicates TObjectType that uses the attribute to represent one of its attributes.

Class

TAttribute

Syntax

```
property Owner: TObjectType;
```

Remarks

Check the value of the Owner property to determine TObjectType that uses the attribute to represent one of its attributes. Applications should not assign the Owner property directly. It is assigned automatically when attribute is created from TOraType.Describe.

 Request Support DAC Forum Provide Feedback

5.11.1.1.2.8 Scale Property

Returns the scale of dtFloat and dtInteger attributes.

Class

TAttribute

Syntax

```
property Scale: Word;
```

Remarks

Use the Scale property to learn the scale of dtFloat and dtInteger attributes.

See Also

- Length

Request Support DAC Forum Provide Feedback

5.11.1.1.2.9 Size Property

Returns the size of an attribute value in external representation.

Class

TAttribute

Syntax

```
property Size: Integer;
```

Remarks

Read Size to learn the size of an attribute value in external representation.

For example:

dtDate	8 (sizeof(TDateTime)
dtFloat	8 (sizeof(Double))

dtInteger	4 (sizeof(Integer))

See Also

- DataSize

 Request Support DAC Forum Provide Feedback

5.11.1.2 TBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types.

For a list of all members of this type, see TBlob members.

Unit

MemData

Syntax

```
TBlob = class(TSharedObject);
```

Remarks

Object TBlob holds large object value for the field and parameter dtBlob, dtMemo, dtWideMemo data types.

Inheritance Hierarchy

TSharedObject

TBlob

See Also

- TMemDataSet.GetBlob

 Request Support DAC Forum Provide Feedback

5.11.1.2.1 Members

TBlob class overview.

Properties

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Sets BLOB value from another TBlob object.
Clear	Deletes the current value in TBlob object.
LoadFromFile	Loads the contents of a file into a TBlob object.
LoadFromStream	Copies the contents of a stream into the TBlob object.
Read	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile	Saves the contents of the TBlob object to a file.
SaveToStream	Copies the contents of a TBlob object to a stream.
Truncate	Sets new TBlob size and discards all data over it.

Write	Stores a raw sequence of bytes into a TBlob object.

 Request Support DAC Forum Provide Feedback

5.11.1.2.2 Properties

Properties of the **TBlob** class.

For a complete list of the **TBlob** class members, see the TBlob Members topic.

Public

Name	Description
AsString	Used to manipulate BLOB value as string.
AsWideString	Used to manipulate BLOB value as Unicode string.
IsUnicode	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of the TBlob value in bytes.

See Also

- TBlob Class
- TBlob Class Members

 Request Support DAC Forum Provide Feedback

5.11.1.2.2.1 AsString Property

Used to manipulate BLOB value as string.

Class

TBlob

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to manipulate BLOB value as string.

See Also

- Assign
- AsWideString

 Request Support DAC Forum Provide Feedback

5.11.1.2.2.2 AsWideString Property

Used to manipulate BLOB value as Unicode string.

Class

TBlob

Syntax

```
property AsWideString: string;
```

Remarks

Use the AsWideString property to manipulate BLOB value as Unicode string.

See Also

- Assign
- AsString

 Request Support DAC Forum Provide Feedback

5.11.1.2.2.3 IsUnicode Property

Gives choice of making TBlob store and process data in Unicode format or not.

Class

TBlob

Syntax

```
property IsUnicode: boolean;
```

Remarks

Set IsUnicode to True if you want TBlob to store and process data in Unicode format.

Note: changing this property raises an exception if TBlob is not empty.

 Request Support DAC Forum Provide Feedback

5.11.1.2.2.4 Size Property

Used to learn the size of the TBlob value in bytes.

Class

TBlob

Syntax

```
property Size: Cardinal;
```

Remarks

Use the Size property to find out the size of the TBlob value in bytes.

 Request Support DAC Forum Provide Feedback

5.11.1.2.3 Methods

Methods of the **TBlob** class.

For a complete list of the **TBlob** class members, see the TBlob Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign	Sets BLOB value from another TBlob object.
Clear	Deletes the current value in TBlob object.
LoadFromFile	Loads the contents of a file into a TBlob object.
LoadFromStream	Copies the contents of a stream into the TBlob object.
Read	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile	Saves the contents of the TBlob object to a file.
SaveToStream	Copies the contents of a TBlob object to a stream.
Truncate	Sets new TBlob size and discards all data over it.
Write	Stores a raw sequence of bytes into a TBlob object.

See Also

- TBlob Class
- TBlob Class Members

5.11.1.2.3.1 Assign Method

Sets BLOB value from another TBlob object.

Class

TBlob

Syntax

```
procedure Assign(Source: TBlob);
```

Parameters

Source

Holds the BLOB from which the value to the current object will be assigned.

Remarks

Call the Assign method to set BLOB value from another TBlob object.

See Also

- LoadFromStream
- AsString
- AsWideString

 Request Support DAC Forum Provide Feedback

5.11.1.2.3.2 Clear Method

Deletes the current value in TBlob object.

Class

TBlob

Syntax

```
procedure Clear; virtual;
```

Remarks

Call the Clear method to delete the current value in TBlob object.

 Request Support DAC Forum Provide Feedback

5.11.1.2.3.3 LoadFromFile Method

Loads the contents of a file into a TBlob object.

Class

TBlob

Syntax

```
procedure LoadFromFile(const FileName: string);
```

Parameters

FileName

Holds the name of the file from which the TBlob value is loaded.

Remarks

Call the LoadFromFile method to load the contents of a file into a TBlob object. Specify the name of the file to load into the field as the value of the FileName parameter.

See Also

- SaveToFile

5.11.1.2.3.4 LoadFromStream Method

Copies the contents of a stream into the TBlob object.

Class

TBlob

Syntax

```
procedure LoadFromStream(Stream: TStream); virtual;
```

Parameters

Stream

Holds the specified stream from which the field's value is copied.

Remarks

Call the LoadFromStream method to copy the contents of a stream into the TBlob object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- SaveToStream

Request Support DAC Forum Provide Feedback

5.11.1.2.3.5 Read Method

Acquires a raw sequence of bytes from the data stored in TBlob.

Class

TBlob

Syntax

```
function Read(Position: Cardinal; Count: Cardinal; Dest: IntPtr): Cardinal; virtual;
```

Parameters

Position

Holds the starting point of the byte sequence.

Count

Holds the size of the sequence in bytes.

Dest

Holds a pointer to the memory area where to store the sequence.

Return Value

Actually read byte count if the sequence crosses object size limit.

Remarks

Call the Read method to acquire a raw sequence of bytes from the data stored in TBlob.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Dest parameter is a pointer to the memory area where to store the sequence.

If the sequence crosses object size limit, function will return actually read byte count.

See Also

- Write

Request Support DAC Forum Provide Feedback

5.11.1.2.3.6 SaveToFile Method

Saves the contents of the TBlob object to a file.

Class

TBlob

Syntax

```
procedure SaveToFile(const FileName: string);
```

Parameters

FileName

Holds a string that contains the name of the file.

Remarks

Call the SaveToFile method to save the contents of the TBlob object to a file. Specify the name of the file as the value of the FileName parameter.

See Also

- LoadFromFile

Request Support DAC Forum Provide Feedback

5.11.1.2.3.7 SaveToStream Method

Copies the contents of a TBlob object to a stream.

Class

TBlob

Syntax

```
procedure SaveToStream(Stream: TStream); virtual;
```

Parameters

Stream
Holds the name of the stream.

Remarks

Call the SaveToStream method to copy the contents of a TBlob object to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

See Also

- LoadFromStream

Request Support DAC Forum Provide Feedback

5.11.1.2.3.8 Truncate Method

Sets new TBlob size and discards all data over it.

Class

TBlob

Syntax

```
procedure Truncate(NewSize: Cardinal); virtual;
```

Parameters

NewSize
Holds the new size of TBlob.

Remarks

Call the Truncate method to set new TBlob size and discard all data over it. If NewSize is greater or equal TBlob.Size, it does nothing.

Request Support DAC Forum Provide Feedback

5.11.1.2.3.9 Write Method

Stores a raw sequence of bytes into a TBlob object.

Class

TBlob

Syntax

```
procedure Write(Position: Cardinal; Count: Cardinal; Source:
IntPtr); virtual;
```

Parameters

Position
Holds the starting point of the byte sequence.

Count
Holds the size of the sequence in bytes.

Source
Holds a pointer to a source memory area.

Remarks

Call the Write method to store a raw sequence of bytes into a TBlob object.

The Position parameter is the starting point of byte sequence which lasts Count number of bytes. The Source parameter is a pointer to a source memory area.

If the value of the Position parameter crosses current size limit of TBlob object, source data will be appended to the object data.

See Also

- Read

5.11.1.3 TCompressedBlob Class

Holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data.

For a list of all members of this type, see TCompressedBlob members.

Unit

MemData

Syntax

```
TCompressedBlob = class(TBlob);
```

Remarks

TCompressedBlob is a descendant of the TBlob class. It holds large object value for field and parameter dtBlob, dtMemo data types and can compress its data. For more information about using BLOB compression see TCustomDADataSet.Options.

Note: Internal compression functions are available in CodeGear Delphi 2007 for Win32, Borland Developer Studio 2006, Borland Delphi 2005, and Borland Delphi 7. To use BLOB compression under Borland Delphi 6 and Borland C++ Builder you should use your own compression functions. To use them set the CompressProc and UncompressProc variables declared in the MemUtils unit.

Example

```
type
   TCompressProc = function(dest: IntPtr; destLen: IntPtr; const source: In
   TUncompressProc = function(dest: IntPtr; destlen: IntPtr; source: IntPtr
var
   CompressProc: TCompressProc;
   UncompressProc: TUncompressProc;
```

Inheritance Hierarchy

TSharedObject

  TBlob

    TCompressedBlob

See Also

- TBlob
- TMemDataSet.GetBlob
- TCustomDADataSet.Options

 Request Support DAC Forum Provide Feedback

Reserved.

5.11.1.3.1 Members

TCompressedBlob class overview.

Properties

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.

Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.

Request Support DAC Forum Provide Feedback

5.11.1.3.2 Properties

Properties of the **TCompressedBlob** class.

For a complete list of the **TCompressedBlob** class members, see the TCompressedBlob Members topic.

Public

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Compressed	Used to indicate if the Blob is compressed.
CompressedSize	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

See Also

- TCompressedBlob Class
- TCompressedBlob Class Members

 Request Support DAC Forum Provide Feedback

5.11.1.3.2.1 Compressed Property

Used to indicate if the Blob is compressed.

Class

TCompressedBlob

Syntax

```
property Compressed: boolean;
```

Remarks

Indicates whether the Blob is compressed. Set this property to True or False to compress or decompress the Blob.

 Request Support DAC Forum Provide Feedback

5.11.1.3.2.2 CompressedSize Property

Used to indicate compressed size of the Blob data.

Class

TCompressedBlob

Syntax

```
property CompressedSize: Cardinal;
```

Remarks

Indicates compressed size of the Blob data.

 Request Support DAC Forum Provide Feedback

5.11.1.4 TDBObject Class

A base class for classes that work with user-defined data types that have attributes.

For a list of all members of this type, see TDBObject members.

Unit

MemData

Syntax

```
TDBObject = class(TSharedObject);
```

Remarks

TDBObject is a base class for classes that work with user-defined data types that have attributes.

Inheritance Hierarchy

TSharedObject

TDBObject

 Request Support DAC Forum Provide Feedback

5.11.1.4.1 Members

TDBObject class overview.

Properties

Name	Description
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a

	TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

 Request Support DAC Forum Provide Feedback

5.11.1.5 TMemData Class

Implements in-memory database.

For a list of all members of this type, see TMemData members.

Unit

MemData

Syntax

```
TMemData = class(TData);
```

Inheritance Hierarchy

TData

TMemData

 Request Support DAC Forum Provide Feedback

5.11.1.5.1 Members

TMemData class overview.

 Request Support DAC Forum Provide Feedback

5.11.1.6 TObjectType Class

Holds description object type and its attributes.

For a list of all members of this type, see TObjectType members.

Unit

MemData

Syntax

```
TObjectType = class(TSharedObject);
```

Remarks

TObjectType holds description object type and its attributes. TObjectType is an ancestor for TOraType.

Inheritance Hierarchy

TSharedObject

TObjectType

See Also

- TOraType

Request Support DAC Forum Provide Feedback

5.11.1.6.1 Members

TObjectType class overview.

Properties

Name	Description
AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or

	dtTable.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Used to learn the size of an object instance.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support DAC Forum Provide Feedback

5.11.1.6.2 Properties

Properties of the **TObjectType** class.

For a complete list of the **TObjectType** class members, see the TObjectType Members topic.

Public

Name	Description
AttributeCount	Used to indicate the number of attributes of type.
Attributes	Used to access separate attributes.
DataType	Used to indicate the type of object dtObject, dtArray or dtTable.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a

	TSharedObject object.
Size	Used to learn the size of an object instance.

See Also

- TObjectType Class
- TObjectType Class Members

5.11.1.6.2.1 AttributeCount Property

Used to indicate the number of attributes of type.

Class

TObjectType

Syntax

```
property AttributeCount: Integer;
```

Remarks

Use the AttributeCount property to determine the number of attributes of type.

5.11.1.6.2.2 Attributes Property(Indexer)

Used to access separate attributes.

Class

TObjectType

Syntax

```
property Attributes[Index: integer]: TAttribute;
```

Parameters

Index
Holds the attribute's ordinal position.

Remarks

Use the Attributes property to access individual attributes. The value of the Index parameter corresponds to the AttributeNo property of TAttribute.

See Also

- TAttribute
- FindAttribute

 Request Support DAC Forum Provide Feedback

5.11.1.6.2.3 DataType Property

Used to indicate the type of object dtObject, dtArray or dtTable.

Class

TObjectType

Syntax

```
property DataType: Word;
```

Remarks

Use the DataType property to determine the type of object dtObject, dtArray or dtTable.

 Request Support DAC Forum Provide Feedback

5.11.1.6.2.4 Size Property

Used to learn the size of an object instance.

Class

TObjectType

Syntax

```
property Size: Integer;
```

Remarks

Use the Size property to find out the size of an object instance. Size is a sum of all attribute sizes.

See Also

- TAttribute.Size

 Request Support DAC Forum Provide Feedback

5.11.1.6.3 Methods

Methods of the **TObjectType** class.

For a complete list of the **TObjectType** class members, see the TObjectType Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

See Also

- TObjectType Class
- TObjectType Class Members

 Request Support DAC Forum Provide Feedback

5.11.1.6.3.1 FindAttribute Method

Indicates whether a specified Attribute component is referenced in the TAttributes object.

Class

TObjectType

Syntax

```
function FindAttribute(const Name: string): TAttribute; virtual;
```

Parameters

Name

Holds the name of the attribute to search for.

Return Value

TAttribute, if an attribute with a matching name was found. Nil Otherwise.

Remarks

Call FindAttribute to determine if a specified Attribute component is referenced in the TAttributes object. Name is the name of the Attribute for which to search. If FindAttribute finds an Attribute with a matching name, it returns the TAttribute. Otherwise it returns nil.

See Also

- TAttribute
- Attributes

5.11.1.7 TSharedObject Class

A base class that allows to simplify memory management for object referenced by several other objects.

For a list of all members of this type, see TSharedObject members.

Unit

MemData

Syntax

```
TSharedObject = class(System.TObject);
```

Remarks

TSharedObject allows to simplify memory management for object referenced by several other objects. TSharedObject holds a count of references to itself. When any object (referer object) is going to use TSharedObject, it calls the TSharedObject.AddRef method. Referer object has to call the TSharedObject.Release method after using TSharedObject.

See Also

- TBlob
- TObjectType

 Request Support DAC Forum Provide Feedback

5.11.1.7.1 Members

TSharedObject class overview.

Properties

Name	Description
RefCount	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

 Request Support DAC Forum Provide Feedback

5.11.1.7.2 Properties

Properties of the **TSharedObject** class.

For a complete list of the **TSharedObject** class members, see the TSharedObject Members topic.

Public

Name	Description
RefCount	Used to return the count of reference to a TSharedObject object.

See Also

- TSharedObject Class
- TSharedObject Class Members

 Request Support DAC Forum Provide Feedback

5.11.1.7.2.1 RefCount Property

Used to return the count of reference to a TSharedObject object.

Class

TSharedObject

Syntax

```
property RefCount: Integer;
```

Remarks

Returns the count of reference to a TSharedObject object.

 Request Support DAC Forum Provide Feedback

5.11.1.7.3 Methods

Methods of the **TSharedObject** class.

For a complete list of the **TSharedObject** class members, see the TSharedObject Members topic.

Public

Name	Description
AddRef	Increments the reference count for the number of references dependent on the TSharedObject object.
Release	Decrements the reference count.

See Also

- TSharedObject Class
- TSharedObject Class Members

Request Support DAC Forum Provide Feedback

5.11.1.7.3.1 AddRef Method

Increments the reference count for the number of references dependent on the TSharedObject object.

Class

TSharedObject

Syntax

```
procedure AddRef;
```

Remarks

Increments the reference count for the number of references dependent on the TSharedObject object.

See Also

- Release

Request Support DAC Forum Provide Feedback

5.11.1.7.3.2 Release Method

Decrements the reference count.

Class

TSharedObject

Syntax

```
procedure Release;
```

Remarks

Call the Release method to decrement the reference count. When RefCount is 1, TSharedObject is deleted from memory.

See Also

- AddRef

Request Support DAC Forum Provide Feedback

5.11.2 Types

Types in the **MemData** unit.

Types

Name	Description
TLocateExOptions	Represents the set of TLocateExOption.
TUpdateRecKinds	Represents the set of TUpdateRecKind.

Request Support DAC Forum Provide Feedback

5.11.2.1 TLocateExOptions Set

Represents the set of TLocateExOption.

Unit

MemData

Syntax

```
TLocateExOptions = set of TLocateExOption;
```


5.11.2.2 TUpdateRecKinds Set

Represents the set of TUpdateRecKind.

Unit

MemData

Syntax

```
TUpdateRecKinds = set of TUpdateRecKind;
```


5.11.3 Enumerations

Enumerations in the **MemData** unit.

Enumerations

Name	Description
TCompressBlobMode	Specifies when the values should be compressed and the way they should be stored.

TConnLostCause	Specifies the cause of the connection loss.
TDANumericType	Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.
TLocateExOption	Allows to set additional search parameters which will be used by the LocateEx method.
TSortType	Specifies a sort type for string fields.
TUpdateRecKind	Indicates records for which the ApplyUpdates method will be performed.

Request Support DAC Forum Provide Feedback

5.11.3.1 TCompressBlobMode Enumeration

Specifies when the values should be compressed and the way they should be stored.

Unit

MemData

Syntax

```
TCompressBlobMode = (cbNone, cbClient, cbServer, cbClientServer);
```

Values

Value	Meaning
cbClient	Values are compressed and stored as compressed data at the client side. Before posting data to the server decompression is performed and data at the server side stored in the original form. Allows to reduce used client memory due to increase access time to field values. The time spent on the opening DataSet and executing Post increases.
cbClientServer	Values are compressed and stored in compressed form. Allows to decrease the volume of used memory at client and server sides. Access time to the field values increases as for cbClient. The time spent on opening DataSet and executing Post

	decreases.**Note:** On using cbServer or cbClientServer data on the server is stored as compressed. Other applications can add records in uncompressed format but can't read and write already compressed data. If compressed BLOB is partially changed by another application (if signature was not changed), DAC will consider its value as NULL.Blob compression is not applied to Memo fields because of possible cutting.
cbNone	Values not compressed. The default value.
cbServer	Values are compressed before passing to the server and store at the server in compressed form. Allows to decrease database size on the server. Access time to the field values does not change. The time spent on opening DataSet and executing Post usually decreases.

Request Support DAC Forum Provide Feedback

5.11.3.2 TConnLostCause Enumeration

Specifies the cause of the connection loss.

Unit

MemData

Syntax

```
TConnLostCause = (clUnknown, clExecute, clOpen, clRefresh, clApply,
clServiceQuery, clTransStart, clConnectionApply, clConnect);
```

Values

Value	Meaning
clApply	Connection loss detected during DataSet.ApplyUpdates (Reconnect/Reexecute possible).
clConnect	Connection loss detected during connection establishing (Reconnect possible).
clConnectionApply	Connection loss detected during Connection.ApplyUpdates (Reconnect/Reexecute possible).
clExecute	Connection loss detected during SQL execution (Reconnect with exception is possible).
clOpen	Connection loss detected during execution of a SELECT statement (Reconnect with exception possible).

clRefresh	Connection loss detected during query opening (Reconnect/ Reexecute possible).
clServiceQuery	Connection loss detected during service information request (Reconnect/Reexecute possible).
clTransStart	Connection loss detected during transaction start (Reconnect/ Reexecute possible). clTransStart has less priority then clConnectionApply.
clUnknown	The connection loss reason is unknown.

Request Support DAC Forum Provide Feedback

5.11.3.3 TDANumericType Enumeration

Specifies the format of storing and representing of the NUMERIC (DECIMAL) fields.

Unit

MemData

Syntax

```
TDANumericType = (ntFloat, ntBCD, ntFmtBCD);
```

Values

Value	Meaning
ntBCD	Data is stored on the client side as currency and represented as TBCDField. This format allows storing data with precision up to 0,0001.
ntFloat	Data stored on the client side is in double format and represented as TFloatField. The default value.
ntFmtBCD	Data is represented as TFMTBCDField. TFMTBCDField gives greater precision and accuracy than TBCDField, but it is slower.

Request Support DAC Forum Provide Feedback

5.11.3.4 TLocateExOption Enumeration

Allows to set additional search parameters which will be used by the LocateEx method.

Unit

MemData

Syntax

```
TLocateExOption = (lxCaseInsensitive, lxPartialKey, lxNearest,
lxNext, lxUp, lxPartialCompare);
```

Values

Value	Meaning
lxCaseInsensitive	Similar to loCaseInsensitive. Key fields and key values are matched without regard to the case.
lxNearest	LocateEx moves the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. For this option to work correctly dataset should be sorted by the fields the search is performed in. If dataset is not sorted, the function may return a line that is not connected with the search condition.
lxNext	LocateEx searches from the current record.
lxPartialCompare	Similar to lxPartialKey, but the difference is that it can process value entries in any position. For example, 'HAM' would match both 'HAMM', 'HAMMER.', and also 'MR HAMMER'.
lxPartialKey	Similar to loPartialKey. Key values can include only a part of the matching key field value. For example, 'HAM' would match both 'HAMM' and 'HAMMER.', but not 'MR HAMMER'.
lxUp	LocateEx searches from the current record to the first record.

5.11.3.5 TSortType Enumeration

Specifies a sort type for string fields.

Unit

MemData

Syntax

```
TSortType = (stCaseSensitive, stCaseInsensitive, stBinary);
```

Values

Value	Meaning
stBinary	Sorting by character ordinal values (this comparison is also case sensitive).
stCaseInsensitive	Sorting without case sensitivity.
stCaseSensitive	Sorting with case sensitivity.

Request Support DAC Forum Provide Feedback

5.11.3.6 TUpdateRecKind Enumeration

Indicates records for which the ApplyUpdates method will be performed.

Unit

MemData

Syntax

```
TUpdateRecKind = (ukUpdate, ukInsert, ukDelete);
```

Values

Value	Meaning
ukDelete	ApplyUpdates will be performed for deleted records.
ukInsert	ApplyUpdates will be performed for inserted records.
ukUpdate	ApplyUpdates will be performed for updated records.

Request Support DAC Forum Provide Feedback

5.12 MemDS

This unit contains implementation of the TMemDataSet class.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.
SendDataSetChangeEventAfterOpen	The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Request Support DAC Forum Provide Feedback

5.12.1 Classes

Classes in the **MemDS** unit.

Classes

Name	Description
TMemDataSet	A base class for working with data and manipulating data in memory.

Request Support DAC Forum Provide Feedback

5.12.1.1 TMemDataSet Class

A base class for working with data and manipulating data in memory.

For a list of all members of this type, see TMemDataSet members.

Unit

MemDS

Syntax

```
TMemDataSet = class(TDataSet);
```

Remarks

TMemDataSet derives from the TDataSet database-engine independent set of properties, events, and methods for working with data and introduces additional techniques to store and manipulate data in memory.

Request Support DAC Forum Provide Feedback

5.12.1.1.1 Members

TMemDataSet class overview.

Properties

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate	Used to prevent implicit update of rows on database server.
Prepared	Determines whether a query is prepared for execution or not.
Ranged	Indicates whether a range is applied to a dataset.
UpdateRecordTypes	Used to indicate the update status for the current record when cached updates are

	enabled.
UpdatesPending	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.
EditRangeEnd	Enables changing the ending value for an existing range.
EditRangeStart	Enables changing the starting value for an existing range.
GetBlob	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.

Prepare	Allocates resources and creates field components for a dataset.
RestoreUpdates	Marks all records in the cache of updates as unapplied.
RevertRecord	Cancels changes made to the current record when cached updates are enabled.
SaveToXML	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange	Sets the starting and ending values of a range, and applies it.
SetRangeEnd	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.12.1.1.2 Properties

Properties of the **TMemDataSet** class.

For a complete list of the **TMemDataSet** class members, see the TMemDataSet Members topic.

Public

Name	Description
CachedUpdates	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive	Specifies the upper and lower boundaries for a range.
LocalConstraints	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate	Used to prevent implicit update of rows on database server.
Prepared	Determines whether a query is prepared for execution or not.
Ranged	Indicates whether a range is

	applied to a dataset.
UpdateRecordTypes	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending	Used to check the status of the cached updates buffer.

See Also

- TMemDataSet Class
- TMemDataSet Class Members

Request Support DAC Forum Provide Feedback

5.12.1.1.2.1 CachedUpdates Property

Used to enable or disable the use of cached updates for a dataset.

Class

TMemDataSet

Syntax

```
property CachedUpdates: boolean default False;
```

Remarks

Use the CachedUpdates property to enable or disable the use of cached updates for a dataset. Setting CachedUpdates to True enables updates to a dataset (such as posting changes, inserting new records, or deleting records) to be stored in an internal cache on the client side instead of being written directly to the dataset's underlying database tables. When changes are completed, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are especially useful for client applications working with remote database servers. Enabling cached updates brings up the following benefits:

- Fewer transactions and shorter transaction times.
- Minimized network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.

The default value is False.

Note: When establishing master/detail relationship the CachedUpdates property of detail dataset works properly only when TDADataSetOptions.LocalMasterDetail is set to True.

See Also

- UpdatesPending
- TMemDataSet.ApplyUpdates
- RestoreUpdates
- CommitUpdates
- CancelUpdates
- UpdateStatus
- TCustomDADataSet.Options

Request Support DAC Forum Provide Feedback

5.12.1.1.2.2 IndexFieldNames Property

Used to get or set the list of fields on which the recordset is sorted.

Class

TMemDataSet

Syntax

```
property IndexFieldNames: string;
```

Remarks

Use the IndexFieldNames property to get or set the list of fields on which the recordset is sorted. Specify the name of each column in IndexFieldNames to use as an index for a table. Column names order is significant. Separate names with semicolons. The specified columns don't need to be indexed. Set IndexFieldNames to an empty string to reset the recordset to the sort order originally used when the recordset's data was first retrieved.

Each field may optionally be followed by the keyword ASC / DESC or CIS / CS / BIN.

Use ASC, DESC keywords to specify a sort order for the field. If one of these keywords is not used, the default sort order for the field is ascending.

Use CIS, CS or BIN keywords to specify the sort type for string fields:

CIS - compare without case sensitivity;

CS - compare with case sensitivity;

BIN - compare by character ordinal values (this comparison is also case sensitive).

If a dataset uses a TCustomDAConnection component, the default value of the sort type depends on the TCustomDAConnection.Options option of the connection. If a dataset does not use a connection (TVirtualTable dataset), the default is CS.

Read IndexFieldNames to determine the field or fields on which the recordset is sorted.

Sorting is performed locally.

Note:

You cannot sort by BLOB fields.

IndexFieldNames cannot be set to True when TCustomDADataSet.UniDirectional=True.

Example

The following procedure illustrates how to set IndexFieldNames in response to a button click:

```
DataSet1.IndexFieldNames := 'LastName ASC CIS; DateDue DESC';
```

5.12.1.1.2.3 KeyExclusive Property

Specifies the upper and lower boundaries for a range.

Class

TMemDataSet

Syntax

```
property KeyExclusive: Boolean;
```

Remarks

Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values.

By default, KeyExclusive is False, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to True.

See Also

- SetRange
- SetRangeEnd
- SetRangeStart

Request Support DAC Forum Provide Feedback

5.12.1.1.2.4 LocalConstraints Property

Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

Class

TMemDataSet

Syntax

```
property LocalConstraints: boolean default True;
```

Remarks

Use the LocalConstraints property to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet. When LocalConstrains is True, TMemDataSet ignores NOT NULL server constraints. It is useful for tables that have fields updated by triggers.

LocalConstraints is obsolete, and is only included for backward compatibility.

The default value is True.

 Request Support DAC Forum Provide Feedback

5.12.1.1.2.5 LocalUpdate Property

Used to prevent implicit update of rows on database server.

Class

TMemDataSet

Syntax

```
property LocalUpdate: boolean default False;
```

Remarks

Set the LocalUpdate property to True to prevent implicit update of rows on database server. Data changes are cached locally in client memory.

 Request Support DAC Forum Provide Feedback

5.12.1.1.2.6 Prepared Property

Determines whether a query is prepared for execution or not.

Class

TMemDataSet

Syntax

```
property Prepared: boolean;
```

Remarks

Check the Prepared property to determine if a query is already prepared for execution. Prepared is True if the query has already been prepared. While queries don't need to be prepared before execution, performance is often boosted if queries are prepared beforehand, particularly if there are parameterized queries that are executed more than once using the same parameter values.

See Also

- Prepare

 Request Support DAC Forum Provide Feedback

5.12.1.1.2.7 Ranged Property

Indicates whether a range is applied to a dataset.

Class

TMemDataSet

Syntax

```
property Ranged: Boolean;
```

Remarks

Use the Ranged property to detect whether a range is applied to a dataset.

See Also

- SetRange
- SetRangeEnd
- SetRangeStart

 Request Support DAC Forum Provide Feedback

5.12.1.1.2.8 UpdateRecordTypes Property

Used to indicate the update status for the current record when cached updates are enabled.

Class

TMemDataSet

Syntax

```
property UpdateRecordTypes: TUpdateRecordTypes default
[rtModified, rtInserted, rtUnmodified];
```

Remarks

Use the UpdateRecordTypes property to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateRecordTypes offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

See Also

- CachedUpdates

Request Support DAC Forum Provide Feedback

5.12.1.1.2.9 UpdatesPending Property

Used to check the status of the cached updates buffer.

Class

TMemDataSet

Syntax

```
property UpdatesPending: boolean;
```

Remarks

Use the UpdatesPending property to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records remaining in local

cache and not yet applied to the database. If UpdatesPending is False, there are no such records in the cache.

See Also

- CachedUpdates

Request Support DAC Forum Provide Feedback

5.12.1.1.3 Methods

Methods of the **TMemDataSet** class.

For a complete list of the **TMemDataSet** class members, see the TMemDataSet Members topic.

Public

Name	Description
ApplyRange	Applies a range to the dataset.
ApplyUpdates	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange	Removes any ranges currently in effect for a dataset.
CancelUpdates	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates	Clears the cached updates buffer.
DeferredPost	Makes permanent changes to the database server.
EditRangeEnd	Enables changing the ending value for an existing range.
EditRangeStart	Enables changing the starting value for an existing range.
GetBlob	Overloaded. Retrieves

	TBlob object for a field or current record when only its name or the field itself is known.
Locate	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare	Allocates resources and creates field components for a dataset.
RestoreUpdates	Marks all records in the cache of updates as unapplied.
RevertRecord	Cancels changes made to the current record when cached updates are enabled.
SaveToXML	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange	Sets the starting and ending values of a range, and applies it.
SetRangeEnd	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare	Frees the resources allocated for a previously prepared query on the server and client sides.

UpdateResult	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TMemDataSet Class
- TMemDataSet Class Members

Request Support DAC Forum Provide Feedback

5.12.1.1.3.1 ApplyRange Method

Applies a range to the dataset.

Class

TMemDataSet

Syntax

```
procedure ApplyRange;
```

Remarks

Call ApplyRange to cause a range established with SetRangeStart and SetRangeEnd, or EditRangeStart and EditRangeEnd, to take effect.

When a range is in effect, only those records that fall within the range are available to the application for viewing and editing.

After a call to ApplyRange, the cursor is left on the first record in the range.

See Also

- CancelRange

- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.2 ApplyUpdates Method

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Overload List

Name	Description
ApplyUpdates	Writes dataset's pending cached updates to a database.
ApplyUpdates(**const** UpdateRecKinds: TUpdateRecKinds)	Writes dataset's pending cached updates of specified records to a database.

 Request Support DAC Forum Provide Feedback

Writes dataset's pending cached updates to a database.

Class

TMemDataSet

Syntax

```
procedure ApplyUpdates; overload; virtual;
```

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

Example

The following procedure illustrates how to apply a dataset's cached updates to a database in response to a button click:

```
procedure ApplyButtonClick(Sender: TObject);
begin
   with MyQuery do
   begin
      Session.StartTransaction;
      try
         ... <Modify data>
         ApplyUpdates; <try to write the updates to the database>
         Session.Commit; <on success, commit the changes>
      except
         RestoreUpdates; <restore update result for applied records>
         Session.Rollback; <on failure, undo the changes>
         raise; <raise the exception to prevent a call to CommitUpdates!>
      end;
      CommitUpdates; <on success, clear the cache>
   end;
end;
```

See Also

- TMemDataSet.CachedUpdates
- TMemDataSet.CancelUpdates
- TMemDataSet.CommitUpdates
- TMemDataSet.UpdateStatus

Writes dataset's pending cached updates of specified records to a database.

Class

TMemDataSet

Syntax

```
procedure ApplyUpdates(const UpdateRecKinds: TUpdateRecKinds);
overload; virtual;
```

Parameters

UpdateRecKinds

Indicates records for which the ApplyUpdates method will be performed.

Remarks

Call the ApplyUpdates method to write a dataset's pending cached updates of specified records to a database. This method passes cached data to the database, but the changes are not committed to the database if there is an active transaction. An application must explicitly call the database component's Commit method to commit the changes to the database if the write is successful, or call the database's Rollback method to undo the changes if there is an error.

Following a successful write to the database, and following a successful call to a connection's Commit method, an application should call the CommitUpdates method to clear the cached update buffer.

Note: The preferred method for updating datasets is to call a connection component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The connection component's ApplyUpdates method takes care of committing and rolling back transactions and clearing the cache when the operation is successful.

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.12.1.1.3.3 CancelRange Method

Removes any ranges currently in effect for a dataset.

Class

TMemDataSet

Syntax

```
procedure CancelRange;
```

Remarks

Call CancelRange to remove a range currently applied to a dataset. Canceling a range reenables access to all records in the dataset.

See Also

- ApplyRange
- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

5.12.1.1.3.4 CancelUpdates Method

Clears all pending cached updates from cache and restores dataset in its prior state.

Class

TMemDataSet

Syntax

```
procedure CancelUpdates;
```

Remarks

Call the CancelUpdates method to clear all pending cached updates from cache and restore dataset in its prior state.

It restores the dataset to the state it was in when the table was opened, cached updates were last enabled, or updates were last successfully applied to the database.

When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- UpdateStatus

Request Support DAC Forum Provide Feedback

5.12.1.1.3.5 CommitUpdates Method

Clears the cached updates buffer.

Class

TMemDataSet

Syntax

```
procedure CommitUpdates;
```

Remarks

Call the CommitUpdates method to clear the cached updates buffer after both a successful call to ApplyUpdates and a database component's Commit method. Clearing the cache after applying updates ensures that the cache is empty except for records that could not be processed and were skipped by the OnUpdateRecord or OnUpdateError event handlers. An application can attempt to modify the records still in cache.

CommitUpdates also checks wether there are pending updates in dataset. And if there are, it

calls ApplyUpdates.

Record modifications made after a call to CommitUpdates repopulate the cached update buffer and require a subsequent call to ApplyUpdates to move them to the database.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- UpdateStatus

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.6 DeferredPost Method

Makes permanent changes to the database server.

Class

TMemDataSet

Syntax

```
procedure DeferredPost;
```

Remarks

Call DeferredPost to make permanent changes to the database server while retaining dataset in its state whether it is dsEdit or dsInsert.

Explicit call to the Cancel method after DeferredPost has been applied does not abandon modifications to a dataset already fixed in database.

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.7 EditRangeEnd Method

Enables changing the ending value for an existing range.

Class

TMemDataSet

Syntax

```
procedure EditRangeEnd;
```

Remarks

Call EditRangeEnd to change the ending value for an existing range.

To specify an end range value, call FieldByName after calling EditRangeEnd.

After assigning a new ending value, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

Request Support DAC Forum Provide Feedback

5.12.1.1.3.8 EditRangeStart Method

Enables changing the starting value for an existing range.

Class

TMemDataSet

Syntax

```
procedure EditRangeStart;
```

Remarks

Call EditRangeStart to change the starting value for an existing range.

To specify a start range value, call FieldByName after calling EditRangeStart.

After assigning a new ending value, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeEnd
- IndexFieldNames
- SetRange
- SetRangeEnd
- SetRangeStart

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.9 GetBlob Method

Retrieves TBlob object for a field or current record when only its name or the field itself is known.

Class

`TMemDataSet`

Overload List

Name	Description
GetBlob(Field: TField)	Retrieves TBlob object for a field or current record when the field itself is known.
GetBlob(**const** FieldName: **string**)	Retrieves TBlob object for a field or current record when its name is known.

 Request Support DAC Forum Provide Feedback

Retrieves TBlob object for a field or current record when the field itself is known.

Class

TMemDataSet

Syntax

```
function GetBlob(Field: TField): TBlob; overload;
```

Parameters

Field

Holds an existing TField object.

Return Value

TBlob object that was retrieved.

Remarks

Call the GetBlob method to retrieve TBlob object for a field or current record when only its name or the field itself is known. FieldName is the name of an existing field. The field should have MEMO or BLOB type.

Request Support DAC Forum Provide Feedback

Retrieves TBlob object for a field or current record when its name is known.

Class

TMemDataSet

Syntax

```
function GetBlob(const FieldName: string): TBlob; overload;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

TBlob object that was retrieved.

Example

```
OraQuery1.GetBlob('Comment').SaveToFile('Comment.txt');
```

See Also

- TBlob

Request Support DAC Forum Provide Feedback

5.12.1.1.3.10 Locate Method

Searches a dataset for a specific record and positions the cursor on it.

Class

TMemDataSet

Overload List

Name	Description
Locate(**const** KeyFields: **array of** TField; **const** KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the specified fields for a specific record and positions cursor on it.
Locate(**const** KeyFields: **string**; **const** KeyValues: variant; Options: TLocateOptions)	Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Request Support DAC Forum Provide Feedback

Searches a dataset by the specified fields for a specific record and positions cursor on it.

Class

TMemDataSet

Syntax

```
function Locate(const KeyFields: array of TField; const
KeyValues: variant; Options: TLocateOptions): boolean;
reintroduce; overload;
```

Parameters

KeyFields

Holds TField objects in which to search.

KeyValues

Holds the variant that specifies the values to match in the key fields.

Options

Holds additional search latitude when searching in string fields.

Return Value

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Request Support DAC Forum Provide Feedback

Searches a dataset by the fields specified by name for a specific record and positions the cursor on it.

Class

TMemDataSet

Syntax

```
function Locate(const KeyFields: string; const KeyValues:
variant; Options: TLocateOptions): boolean; overload; override;
```

Parameters

KeyFields

Holds a semicolon-delimited list of field names in which to search.

KeyValues

Holds the variant that specifies the values to match in the key fields.

Options

Holds additional search latitude when searching in string fields.

Return Value

True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

Remarks

Call the Locate method to search a dataset for a specific record and position cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify

multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. An example is provided below.

Options is a set that optionally specifies additional search latitude when searching in string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes this record the current one. Otherwise it returns False.

The Locate function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.

Example

An example of specifying multiple search values:

```
with CustTable do
   Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
      '408-431-1000']), [loPartialKey]);
```

See Also

- TMemDataSet.IndexFieldNames
- TMemDataSet.LocateEx

5.12.1.1.3.11 LocateEx Method

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.

Class

TMemDataSet

Overload List

Name	Description

LocateEx(**const** KeyFields: **array of** TField; **const** KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.
LocateEx(**const** KeyFields: **string**; **const** KeyValues: variant; Options: TLocateExOptions)	Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified fields.

Class

TMemDataSet

Syntax

```
function LocateEx(const KeyFields: array of TField; const
KeyValues: variant; Options: TLocateExOptions): boolean; overload;
```

Parameters

KeyFields

Holds TField objects to search in.

KeyValues

Holds the values of the fields to search for.

Options

Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet by the specified field names.

Class

TMemDataSet

Syntax

```
function LocateEx(const KeyFields: string; const KeyValues:
variant; Options: TLocateExOptions): boolean; overload;
```

Parameters

KeyFields

Holds the fields to search in.

KeyValues

Holds the values of the fields to search for.

Options

Holds additional search parameters which will be used by the LocateEx method.

Return Value

True, if a matching record was found. Otherwise returns False.

Remarks

Call the LocateEx method when you need some features not to be included to the TMemDataSet.Locate method of TDataSet.

LocateEx returns True if it finds a matching record, and makes that record the current one. Otherwise LocateEx returns False.

The LocateEx function works faster when dataset is locally sorted on the KeyFields fields. Local dataset sorting can be set with the TMemDataSet.IndexFieldNames property.

Note: Please add the MemData unit to the "uses" list to use the TLocalExOption enumeration.

See Also

- TMemDataSet.IndexFieldNames
- TMemDataSet.Locate

5.12.1.1.3.12 Prepare Method

Allocates resources and creates field components for a dataset.

Class

TMemDataSet

Syntax

```
procedure Prepare; virtual;
```

Remarks

Call the Prepare method to allocate resources and create field components for a dataset. The Prepare method is called automatically by the Open method if dataset is not prepared. To learn whether dataset is prepared or not use the Prepared property.

The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

The Prepare method is called automatically by the Open method if dataset is not prepared.

See Also

- Prepared
- UnPrepare

5.12.1.1.3.13 RestoreUpdates Method

Marks all records in the cache of updates as unapplied.

Class

TMemDataSet

Syntax

```
procedure RestoreUpdates;
```

Remarks

Call the RestoreUpdates method to return the cache of updates to its state before calling ApplyUpdates. RestoreUpdates marks all records in the cache of updates as unapplied. It is useful when ApplyUpdates fails.

See Also

- CachedUpdates
- TMemDataSet.ApplyUpdates
- CancelUpdates
- UpdateStatus

Request Support DAC Forum Provide Feedback

5.12.1.1.3.14 RevertRecord Method

Cancels changes made to the current record when cached updates are enabled.

Class

TMemDataSet

Syntax

```
procedure RevertRecord;
```

Remarks

Call the RevertRecord method to undo changes made to the current record when cached updates are enabled.

See Also

- CachedUpdates
- CancelUpdates

Request Support DAC Forum Provide Feedback

5.12.1.1.3.15 SaveToXML Method

Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.

Class

TMemDataSet

Overload List

Name	Description
SaveToXML(Destination: TStream)	Saves the current dataset data to a stream in the XML format compatible with ADO format.
SaveToXML(**const** FileName: **string**)	Saves the current dataset data to a file in the XML format compatible with ADO format.

Request Support DAC Forum Provide Feedback

Saves the current dataset data to a stream in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

```
procedure SaveToXML(Destination: TStream); overload;
```

Parameters

Destination
Holds a TStream object.

Remarks

Call the SaveToXML method to save the current dataset data to a file or a stream in the XML format compatible with ADO format.

If the destination file already exists, it is overwritten. It remains open from the first call to SaveToXML until the dataset is closed. This file can be read by other applications while it is

opened, but they cannot write to the file.

When saving data to a stream, a TStream object must be created and its position must be set in a preferable value.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream

 Request Support DAC Forum Provide Feedback

Saves the current dataset data to a file in the XML format compatible with ADO format.

Class

TMemDataSet

Syntax

```
procedure SaveToXML(const FileName: string); overload;
```

Parameters

FileName

Holds the name of a destination file.

See Also

- TVirtualTable.LoadFromFile
- TVirtualTable.LoadFromStream

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.16 SetRange Method

Sets the starting and ending values of a range, and applies it.

Class

TMemDataSet

Syntax

```
procedure SetRange(const StartValues: array of System.TVarRec;
const EndValues: array of System.TVarRec; StartExlusive: Boolean
= False; EndExclusive: Boolean = False);
```

Parameters

StartValues

Indicates the field values that designate the first record in the range. In C++, StartValues_Size is the index of the last value in the StartValues array.

EndValues

Indicates the field values that designate the last record in the range. In C++, EndValues_Size is the index of the last value in the EndValues array.

StartExlusive

lindicates the upper and lower boundaries of the start range.

EndExclusive

Indicates the upper and lower boundaries of the end range.

Remarks

Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently specified range, if any.

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

- 1. Puts the dataset into dsSetKey state.
- 2. Erases any previously specified starting range values and ending range values.
- 3. Sets the start and end range values.
- 4. Applies the range to the dataset.

After a call to SetRange, the cursor is left on the first record in the range.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are ignored when performing a search.

See Also

- ApplyRange
- CancelRange

- EditRangeEnd
- EditRangeStart
- IndexFieldNames
- KeyExclusive
- SetRangeEnd
- SetRangeStart

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.17 SetRangeEnd Method

Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeEnd;
```

Remarks

Call SetRangeEnd to put the dataset into dsSetKey state, erase any previous end range values, and set them to NULL.

Subsequent field assignments made with FieldByName specify the actual set of ending values for a range.

After assigning end-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeStart
- IndexFieldNames

- SetRange
- SetRangeStart

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.18 SetRangeStart Method

Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.

Class

TMemDataSet

Syntax

```
procedure SetRangeStart;
```

Remarks

Call SetRangeStart to put the dataset into dsSetKey state, erase any previous start range values, and set them to NULL.

Subsequent field assignments to FieldByName specify the actual set of starting values for a range.

After assigning start-range values, call ApplyRange to activate the modified range.

See Also

- ApplyRange
- CancelRange
- EditRangeStart
- IndexFieldNames
- SetRange
- SetRangeEnd

 Request Support DAC Forum Provide Feedback

5.12.1.1.3.19 UnPrepare Method

Frees the resources allocated for a previously prepared query on the server and client sides.

Class

TMemDataSet

Syntax

```
procedure UnPrepare; virtual;
```

Remarks

Call the UnPrepare method to free the resources allocated for a previously prepared query on the server and client sides.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

See Also

- Prepare

Request Support DAC Forum Provide Feedback

5.12.1.1.3.20 UpdateResult Method

Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.

Class

TMemDataSet

Syntax

```
function UpdateResult: TUpdateAction;
```

Return Value

a value of the TUpdateAction enumeration.

Remarks

Call the UpdateResult method to read the status of the latest call to the ApplyUpdates method while cached updates are enabled. UpdateResult reflects updates made on the records that have been edited, inserted, or deleted.

UpdateResult works on the record by record basis and is applicable to the current record only.

See Also

- CachedUpdates

Request Support DAC Forum Provide Feedback

5.12.1.1.3.21 UpdateStatus Method

Indicates the current update status for the dataset when cached updates are enabled.

Class

TMemDataSet

Syntax

```
function UpdateStatus: TUpdateStatus; override;
```

Return Value

a value of the TUpdateStatus enumeration.

Remarks

Call the UpdateStatus method to determine the current update status for the dataset when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of the dataset.

See Also

- CachedUpdates

Request Support DAC Forum Provide Feedback

5.12.1.1.4 Events

Events of the **TMemDataSet** class.

For a complete list of the **TMemDataSet** class members, see the TMemDataSet Members topic.

Public

Name	Description
OnUpdateError	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord	Occurs when a single update component can not handle the updates.

See Also

- TMemDataSet Class
- TMemDataSet Class Members

 Request Support DAC Forum Provide Feedback

5.12.1.1.4.1 OnUpdateError Event

Occurs when an exception is generated while cached updates are applied to a database.

Class

TMemDataSet

Syntax

```
property OnUpdateError: TUpdateErrorEvent;
```

Remarks

Write the OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

E is a pointer to an EDatabaseError object from which application can extract an error

message and the actual cause of the error condition. The OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind describes the type of update that generated the error.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler.

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate error conditions and set TField.NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

Note: If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the context of a try...except block, an error message is displayed. If the OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is displayed twice. To prevent redisplay, set UpdateAction to uaAbort in the error handler.

See Also

- CachedUpdates

5.12.1.1.4.2 OnUpdateRecord Event

Occurs when a single update component can not handle the updates.

Class

TMemDataSet

Syntax

```
property OnUpdateRecord: TUpdateRecordEvent;
```

Remarks

Write the OnUpdateRecord event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over

parameter substitution in update components.

UpdateKind describes the type of update to perform.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

See Also

- CachedUpdates

Request Support DAC Forum Provide Feedback

5.12.2 Variables

Variables in the **MemDS** unit.

Variables

Name	Description
DoNotRaiseExcetionOnUaFail	An exception will be raised if the value of the UpdateAction parameter is uaFail.
SendDataSetChangeEventAfterOpen	The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Request Support DAC Forum Provide Feedback

5.12.2.1 DoNotRaiseExcetionOnUaFail Variable

An exception will be raised if the value of the UpdateAction parameter is uaFail.

Unit

MemDS

Syntax

```
DoNotRaiseExcetionOnUaFail: boolean = False;
```

Remarks

Starting with ODAC 6.20.0.12, if the OnUpdateRecord event handler sets the UpdateAction parameter to uaFail, an exception is raised. The default value of UpdateAction is uaFail. So, the exception will be raised when the value of this parameter is left unchanged.

To restore the old behaviour, set DoNotRaiseExcetionOnUaFail to True.

Request Support DAC Forum Provide Feedback

5.12.2.2 SendDataSetChangeEventAfterOpen Variable

The DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids.

Unit

MemDS

Syntax

```
SendDataSetChangeEventAfterOpen: boolean = True;
```

Remarks

Starting with ODAC 6.20.0.11, the DataSetChange event is sent after a dataset gets open. It was necessary to fix a problem with disappeared vertical scrollbar in some types of DB-aware grids. This problem appears only under Windows XP when visual styles are enabled.

To disable sending this event, change the value of this variable to False.

Request Support DAC Forum Provide Feedback

5.13 OdacVcl

This unit contains the visual constituent of ODAC.

Classes

Name	Description
TConnectDialog	A component providing a dialog box for a user to supply login information.

 Request Support DAC Forum Provide Feedback

5.13.1 Classes

Classes in the **OdacVcl** unit.

Classes

Name	Description
TConnectDialog	A component providing a dialog box for a user to supply login information.

 Request Support DAC Forum Provide Feedback

5.13.1.1 TConnectDialog Class

A component providing a dialog box for a user to supply login information.

For a list of all members of this type, see TConnectDialog members.

Unit

odacvcl

Syntax

```
TConnectDialog = class(TCustomConnectDialog);
```

Remarks

TConnectDialog component is a direct descendent of TCustomConnectDialog class. Use TConnectDialog to provide a dialog box for a user to supply username, password and server name. You may want to customize appearance of the dialog box using the properties of this class.

Inheritance Hierarchy

TCustomConnectDialog

TConnectDialog

See Also

- TCustomDAConnection.ConnectDialog

Request Support DAC Forum Provide Feedback

5.13.1.1.1 Members

TConnectDialog class overview.

Properties

Name	Description
CancelButton (inherited from **TCustomConnectDialog**)	Used to specify the label for the Cancel button.
Caption (inherited from **TCustomConnectDialog**)	Used to set the caption of dialog box.
ConnectButton (inherited from **TCustomConnectDialog**)	Used to specify the label for the Connect button.
DialogClass (inherited from **TCustomConnectDialog**)	Used to specify the class of the form that will be displayed to enter login information.
LabelSet (inherited from **TCustomConnectDialog**)	Used to set the language of buttons and labels captions.
PasswordLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for password edit.
ReadAliases	Used to specify where the TConnectDialog object should acquire the names of database instances.

Retries (inherited from **TCustomConnectDialog**)	Used to indicate the number of retries of failed connections.
SavePassword (inherited from **TCustomConnectDialog**)	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for the server name edit.
Session	Shows what TOraSession component uses TConnectDialog object.
StoreLogInfo (inherited from **TCustomConnectDialog**)	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for username edit.

Methods

Name	Description
Execute (inherited from **TCustomConnectDialog**)	Displays the connect dialog and calls the connection's Connect method when user clicks the Connect button.
GetServerList (inherited from **TCustomConnectDialog**)	Retrieves a list of available server names.

Request Support DAC Forum Provide Feedback

5.13.1.1.2 Properties

Properties of the **TConnectDialog** class.

For a complete list of the **TConnectDialog** class members, see the TConnectDialog Members topic.

Public

Name	Description
CancelButton (inherited from **TCustomConnectDialog**)	Used to specify the label for the Cancel button.
Caption (inherited from **TCustomConnectDialog**)	Used to set the caption of dialog box.
ConnectButton (inherited from **TCustomConnectDialog**)	Used to specify the label for the Connect button.
DialogClass (inherited from **TCustomConnectDialog**)	Used to specify the class of the form that will be displayed to enter login information.
LabelSet (inherited from **TCustomConnectDialog**)	Used to set the language of buttons and labels captions.
PasswordLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for password edit.
Retries (inherited from **TCustomConnectDialog**)	Used to indicate the number of retries of failed connections.
SavePassword (inherited from **TCustomConnectDialog**)	Used for the password to be displayed in ConnectDialog in asterisks.
ServerLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for the server name edit.
Session	Shows what TOraSession component uses TConnectDialog object.
StoreLogInfo (inherited from **TCustomConnectDialog**)	Used to specify whether the login information should be kept in system registry after a connection was established.
UsernameLabel (inherited from **TCustomConnectDialog**)	Used to specify a prompt for username edit.

Published

Name	Description
ReadAliases	Used to specify where the TConnectDialog object should acquire the names of database instances.

See Also

- TConnectDialog Class
- TConnectDialog Class Members

 Request Support DAC Forum Provide Feedback

5.13.1.1.2.1 ReadAliases Property

Used to specify where the TConnectDialog object should acquire the names of database instances.

Class

TConnectDialog

Syntax

```
property ReadAliases: boolean default False;
```

Remarks

Use ReadAliases property to specify whether the TConnectDialog object should acquire the names of database instances from TNSNAMES.ORA configuration file found in the Oracle home directory or read them from Odac registry entries.

Set this property to False to make ODAC read aliases from registry.

The default value is False.

Note: ODAC relies on valid local Oracle home directory structure if ReadAliases property is set to True.

Description

T:Devart.Odac.Units.OdacVcl

 Request Support DAC Forum Provide Feedback

5.13.1.1.2.2 Session Property

Shows what TOraSession component uses TConnectDialog object.

Class

TConnectDialog

Syntax

```
property Session: TOraSession;
```

Remarks

Read Session property to learn what TOraSession component uses TConnectDialog object.

This property is read-only.

See Also

- TCustomDAConnection.ConnectDialog

5.14 Ora

This unit contains main components of ODAC.

Classes

Name	Description
TBFileField	A class representing BFile field in dataset.
TCursorField	A class representing REF CURSOR field in dataset.
TCustomOraQuery	A base class defining functionality for descendent classes which access database using SQL statements.
TOraChangeNotification	A component for keeping information in local dataset up-to-date through receiving notifications.

TOraChangeNotificationOptions	Used to specify how query notifications will be generated.
TOraConnectionSSLOptions	A class for setting up the SSL options.
TOraDataSet	A class defining the Oracle functionality for a dataset.
TOraDataSetField	A class providing access to Oracle nested datasets.
TOraDataSetOptions	This class allows setting up the behaviour of the TOraDataSet class.
TOraDataSetOptionsDS	This class allows setting up the behaviour of the TOraDaatSet class (this property is obsolete).
TOraDataSource	TOraDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.
TOraEncryptor	The class that performs encrypting and decrypting of data.
TOraIntervalField	A class providing access to the Oracle interval fields.
TOraMetaData	A component for obtaining metainformation about database objects from the server.
TOraNestedTable	A component for controlling nested table data.
TOraNumberField	A class providing access to the Oracle number fields.
TOraParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TOraParams	Used to control TOraParam objects.
TOraPoolingOptions	This class allows setting up the behaviour of the connecton pool.

TOraQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TOraReferenceField	A class representing an Oracle REF field in a dataset.
TOraSession	A component for maintaining connection to an Oracle database.
TOraSessionOptions	This class allows setting up the behaviour of the TOraSession class.
TOraSQL	A component for executing SQL statements and calling stored procedures on the database server.
TOraStoredProc	A component for accessing and executing stored procedures and functions.
TOraTimeStampField	A class providing access to the Oracle timestamp fields.
TOraTrace	A component allowing starting and stopping a SQL trace for a specified session. This component provides access to the DBMS_TRACE package.
TOraUpdateSQL	A component for tuning update operations for the DataSet component.
TOraXMLField	A class providing access to the Oracle SYS.XMLTYPE objects.

Types

Name	Description
TConnectChangeEvent	This type is used for the TOraSession.OnConnectChange event.
TFailoverEvent	This Type is used for the TOraSession.OnFailover event.

TOraChangeNotificationEvent	This type is used for the TOraChangeNotification.OnChange event.
TPlSqlTraceMode	Specifies the level of PL/SQL trace.
TSqlTraceMode	Specifies the level of SQL trace statistics level.

Enumerations

Name	Description
TFailoverState	Indicates the failover state.
TFailoverType	Specifies the failover type.
TOraIsolationLevel	Specifies the way the transactions containing database modifications are handled.
TRefreshMode	Defines when to refresh an editing record.
TSequenceMode	Specifies the method used internally to generate sequenced field.

Variables

Name	Description
DefSession	Read this variable to get pointer to default session object. Same as DefaultSession function.
OraQueryCompatibilityMode	All TOraQuery components in project become editable, and can be modified by the end users.
Sessions	Holds pointers to all TOraSession objects of an application.
UseDefSession	When set to True enables TOraDataSet and TOraSQL components to use default session if they are not attached to any session.

5.14.1 Classes

Classes in the **Ora** unit.

Classes

Name	Description
TBFileField	A class representing BFile field in dataset.
TCursorField	A class representing REF CURSOR field in dataset.
TCustomOraQuery	A base class defining functionality for descendent classes which access database using SQL statements.
TOraChangeNotification	A component for keeping information in local dataset up-to-date through receiving notifications.
TOraChangeNotificationOptions	Used to specify how query notifications will be generated.
TOraConnectionSSLOptions	A class for setting up the SSL options.
TOraDataSet	A class defining the Oracle functionality for a dataset.
TOraDataSetField	A class providing access to Oracle nested datasets.
TOraDataSetOptions	This class allows setting up the behaviour of the TOraDataSet class.
TOraDataSetOptionsDS	This class allows setting up the behaviour of the TOraDaatSet class (this property is obsolete).
TOraDataSource	TOraDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.

TOraEncryptor	The class that performs encrypting and decrypting of data.
TOraIntervalField	A class providing access to the Oracle interval fields.
TOraMetaData	A component for obtaining metainformation about database objects from the server.
TOraNestedTable	A component for controlling nested table data.
TOraNumberField	A class providing access to the Oracle number fields.
TOraParam	A class that is used to set the values of individual parameters passed with queries or stored procedures.
TOraParams	Used to control TOraParam objects.
TOraPoolingOptions	This class allows setting up the behaviour of the connecton pool.
TOraQuery	A component for executing queries and operating record sets. It also provides flexible way to update data.
TOraReferenceField	A class representing an Oracle REF field in a dataset.
TOraSession	A component for maintaining connection to an Oracle database.
TOraSessionOptions	This class allows setting up the behaviour of the TOraSession class.
TOraSQL	A component for executing SQL statements and calling stored procedures on the database server.
TOraStoredProc	A component for accessing and executing stored procedures and functions.
TOraTimeStampField	A class providing access to

	the Oracle timestamp fields.
TOraTrace	A component allowing starting and stopping a SQL trace for a specified session. This component provides access to the DBMS_TRACE package.
TOraUpdateSQL	A component for tuning update operations for the DataSet component.
TOraXMLField	A class providing access to the Oracle SYS.XMLTYPE objects.

Request Support DAC Forum Provide Feedback

5.14.1.1 TBFileField Class

A class representing BFile field in dataset.

For a list of all members of this type, see TBFileField members.

Unit

Ora

Syntax

```
TBFileField = class(TBlobField);
```

Remarks

TBFileField object represents BFile field in dataset.

The BFile datatype provides access to file LOBs that are stored in file systems outside an Oracle database. Oracle 8 currently supports access to binary files, or BFILEs. The BFILE datatype allows read-only support of large binary files; you cannot modify a file through Oracle.

TBFileField holds a TOraFile object. To get it use the AsFile property.

As a descendent of TField, TBFileField inherits many properties, methods, and events that are useful for managing the value and properties of a field in the database.

See Also

- TOraFile

5.14.1.1.1 Members

TBFileField class overview.

Properties

Name	Description
AsFile	Returns a TOraFile object.
AutoRefresh	Used to refresh content of BFile when FileDir or FileName is being changed.
BlobType	Indicates the type of BLOB field.
Exists	Returns True when a file associated with BFile exists, False otherwise.
FileDir	Used to indicate the directory alias where BFile is stored.
FileName	Indicates the name of a file associated with BFile.

Methods

Name	Description
Refresh	Reloads BFile.

5.14.1.1.2 Properties

Properties of the **TBFileField** class.

For a complete list of the **TBFileField** class members, see the TBFileField Members topic.

Public

Name	Description
AsFile	Returns a TOraFile object.
Exists	Returns True when a file associated with BFile exists, False otherwise.
FileDir	Used to indicate the directory alias where BFile is stored.
FileName	Indicates the name of a file associated with BFile.

Published

Name	Description
AutoRefresh	Used to refresh content of BFile when FileDir or FileName is being changed.
BlobType	Indicates the type of BLOB field.

See Also

- TBFileField Class
- TBFileField Class Members

Request Support DAC Forum Provide Feedback

5.14.1.1.2.1 AsFile Property

Returns a TOraFile object.

Class

TBFileField

Syntax

```
property AsFile: TOraFile;
```

Remarks

Returns a TOraFile object. Later you can open TOraDataSet once.

See Also

- TOraFile

Request Support DAC Forum Provide Feedback

5.14.1.1.2.2 AutoRefresh Property

Used to refresh content of BFile when FileDir or FileName is being changed.

Class

TBFileField

Syntax

```
property AutoRefresh: boolean default True;
```

Remarks

When AutoRefresh is True, TBFileField will refresh content of BFile when FileDir or FileName is changed.

The default value is True.

Request Support DAC Forum Provide Feedback

5.14.1.1.2.3 BlobType Property

Indicates the type of BLOB field.

Class

TBFileField

Syntax

```
property BlobType: TBlobType;
```

Remarks

Indicates the type of BLOB field. For TBFileField is always equal to ftBlob.

Request Support DAC Forum Provide Feedback

5.14.1.1.2.4 Exists Property

Returns True when a file associated with BFile exists, False otherwise.

Class

TBFileField

Syntax

```
property Exists: boolean;
```

Remarks

Exists returns True when a file associated with BFile exists, False otherwise.

Example

```
if not TBFileField(DataSet.FieldByName('Value')).Exists then
   St:= St + '   (NoExist)';
```


Request Support DAC Forum Provide Feedback

5.14.1.1.2.5 FileDir Property

Used to indicate the directory alias where BFile is stored.

Class

TBFileField

Syntax

```
property FileDir: string;
```

Remarks

Use the FileDir property to determine the directory alias where BFile is stored.

To create a directory alias use CREATE DIRECTORY.

Example

```
edFileDir.Text:= TBFileField(OraQuery.FieldByName('Value')).FileDir;
```


Request Support DAC Forum Provide Feedback

5.14.1.1.2.6 FileName Property

Indicates the name of a file associated with BFile.

Class

TBFileField

Syntax

```
property FileName: string;
```

Remarks

Use the FileName property to determine the name of a file associated with BFile.

Example

```
edFileName.Text := BFileField(OraQuery.FieldByName('Value')).FileName;
```


Request Support DAC Forum Provide Feedback

5.14.1.1.3 Methods

Methods of the **TBFileField** class.

For a complete list of the **TBFileField** class members, see the TBFileField Members topic.

Public

Name	Description
Refresh	Reloads BFile.

See Also

- TBFileField Class
- TBFileField Class Members

5.14.1.1.3.1 Refresh Method

Reloads BFile.

Class

TBFileField

Syntax

```
procedure Refresh;
```

Remarks

Call the Refresh procedure to reload BFile.

5.14.1.2 TCursorField Class

A class representing REF CURSOR field in dataset.

For a list of all members of this type, see TCursorField members.

Unit

Ora

Syntax

```
TCursorField = class(TDACursorField);
```

Remarks

A TCursorField object represents REF CURSOR field in dataset.

TCursorField holds a TOraCursor object. To get it use the AsCursor property.

As a descendent of TField, TCursorField inherits many properties, methods, and events that are useful for managing the value and properties of a field in a database.

Inheritance Hierarchy

TDACursorField

TCursorField

See Also

- TOraCursor
- TOraDataSet.Cursor

 Request Support DAC Forum Provide Feedback

5.14.1.2.1 Members

TCursorField class overview.

Properties

Name	Description
AsCursor	Returns a TOraCursor object you can assign to the Cursor property of TOraDataSet.

 Request Support DAC Forum Provide Feedback

5.14.1.2.2 Properties

Properties of the **TCursorField** class.

For a complete list of the **TCursorField** class members, see the TCursorField Members topic.

Public

Name	Description

AsCursor	Returns a TOraCursor object you can assign to the Cursor property of TOraDataSet.

See Also

- TCursorField Class
- TCursorField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.2.2.1 AsCursor Property

Returns a TOraCursor object you can assign to the Cursor property of TOraDataSet.

Class

TCursorField

Syntax

```
property AsCursor: TOraCursor;
```

Remarks

Returns a TOraCursor object which you can assign to the Cursor property of TOraDataSet. Later you can open TOraDataSet once.

See Also

- TOraDataSet.Cursor

 Request Support DAC Forum Provide Feedback

5.14.1.3 TCustomOraQuery Class

A base class defining functionality for descendent classes which access database using SQL statements.

For a list of all members of this type, see TCustomOraQuery members.

Unit

Ora

Syntax

```
TCustomOraQuery = class(TOraDataSet);
```

Remarks

TCustomOraQuery is a base class that defines functionality for descendent classes which access database using SQL statements. Applications never use TCustomOraQuery objects directly. Instead they use descendants of TCustomOraQuery, such as TOraQuery, TSmartQuery, TOraStoredProc and TOraTable.

TCustomOraQuery implements functionality for an insertion, deletion, and update of a record by dynamically generated SQL statements. It offers such features as automatic blocking of records, checking records before edit, refreshing records after post.

To modify records of TCustomOraQuery SELECT statement in the SQL property should retrieve ROWID of updating table. When the KeyFields property is modified, TCustomOraQuery is updated too. TCustomOraQuery can update only one Oracle table. Updating table is defined by the UpdatingTable property or used by the first table in the FROM clause.

SQLInsert, SQLDelete, SQLUpdate, SQLLock, SQLRefresh properties support automatic binding of parameters which have names identical to the fields captions. To retrieve the value of a field as it was before operation, use the field name with 'OLD_'. This is particularly useful when doing field comparisons in the WHERE clause of a statement. Use the TCustomDADataSet.BeforeUpdateExecute event to assign the value to additional parameters and the TCustomDADataSet.AfterUpdateExecute event for reading them.

TCustomOraQuery is read-only when none of the SQLInsert, SQLDelete, SQLUpdate properties are defined.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet

TCustomOraQuery

See Also

- TOraDataSet
- TOraQuery
- TSmartQuery
- TOraStoredProc
- TOraTable

Request Support DAC Forum Provide Feedback

5.14.1.3.1 Members

TCustomOraQuery class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query

Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of

	fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode (inherited from **TOraDataSet**)	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.

MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.

SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.

StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.

CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that

	matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.

GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.

RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

5.14.1.4 TOraChangeNotification Class

A component for keeping information in local dataset up-to-date through receiving notifications.

For a list of all members of this type, see TOraChangeNotification members.

Unit

Ora

Syntax

```
TOraChangeNotification = class(TComponent);
```

Remarks

The TOraChangeNotification component is used to register queries with the database and receive notifications in response to DML or DDL changes on the objects associated with queries. The notifications are published by database when the DML or DDL transaction commits.

See Also

- TOraChangeNotification Component

Request Support DAC Forum Provide Feedback

5.14.1.4.1 Members

TOraChangeNotification class overview.

Properties

Name	Description
Active	Indicates whether there is an active dataset connected with the TOraChangeNotification component.
Enabled	Used to enable or disable using change notification.
Operations	Used to be notified of the particular operations execution.
Persistent	Used to store notifications in a database persistently.
TimeOut	Indicates the interval for a notification to remain active.

Methods

Name	Description
RemoveRegistration	Removes the change notification registration for all open datasets, connected

	with the TOraChangeNotification component.

Events

Name	Description
OnChange	Occurs when data in one of the associated datasets was changed on the server.

Request Support DAC Forum Provide Feedback

5.14.1.4.2 Properties

Properties of the **TOraChangeNotification** class.

For a complete list of the **TOraChangeNotification** class members, see the TOraChangeNotification Members topic.

Public

Name	Description
Active	Indicates whether there is an active dataset connected with the TOraChangeNotification component.
Persistent	Used to store notifications in a database persistently.

Published

Name	Description
Enabled	Used to enable or disable using change notification.
Operations	Used to be notified of the particular operations execution.
TimeOut	Indicates the interval for a notification to remain active.

See Also

- TOraChangeNotification Class
- TOraChangeNotification Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.4.2.1 Active Property

Indicates whether there is an active dataset connected with the TOraChangeNotification component.

Class

TOraChangeNotification

Syntax

```
property Active: boolean;
```

 Request Support DAC Forum Provide Feedback

5.14.1.4.2.2 Enabled Property

Used to enable or disable using change notification.

Class

TOraChangeNotification

Syntax

```
property Enabled: boolean default True;
```

Remarks

Set the Enabled property to False to disable change notification for all datasets connected to the TOraChangeNotification component. Setting this property to True allows datasets, connected to the TOraChangeNotification component, to use change notification.

 Request Support DAC Forum Provide Feedback

5.14.1.4.2.3 Operations Property

Used to be notified of the particular operations execution.

Class

TOraChangeNotification

Syntax

```
property Operations: TChangeNotifyDMLOperations default
[cnoInsert, cnoUpdate, cnoDelete];
```

Remarks

Set the Operations property to provide a notification when particular operations are being executed.

 Request Support DAC Forum Provide Feedback

5.14.1.4.2.4 Persistent Property

Used to store notifications in a database persistently.

Class

TOraChangeNotification

Syntax

```
property Persistent: boolean;
```

Remarks

If True, notifications will be stored persistently in a database and would not be lost if server instance crashes after generating notifications, and they would be sent after Oracle server restart.

 Request Support DAC Forum Provide Feedback

5.14.1.4.2.5 TimeOut Property

Indicates the interval for a notification to remain active.

Class

TOraChangeNotification

Syntax

```
property TimeOut: integer default 0;
```

Remarks

Set the TimeOut property to determine time interval in seconds, after which the notification registration will expire.

The minimum value is 1 second, maximum is 2^31-1 seconds.

Request Support DAC Forum Provide Feedback

5.14.1.4.3 Methods

Methods of the **TOraChangeNotification** class.

For a complete list of the **TOraChangeNotification** class members, see the TOraChangeNotification Members topic.

Public

Name	Description
RemoveRegistration	Removes the change notification registration for all open datasets, connected with the TOraChangeNotification component.

See Also

- TOraChangeNotification Class
- TOraChangeNotification Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.4.3.1 RemoveRegistration Method

Removes the change notification registration for all open datasets, connected with the TOraChangeNotification component.

Class

TOraChangeNotification

Syntax

```
procedure RemoveRegistration(Session: TOraSession);
```

Parameters

Session

Remarks

Use the RemoveRegistration method to remove the change notification registration for all open datasets, connected with the TOraChangeNotification component. To register the change notification again, reopen all required datasets.

 Request Support DAC Forum Provide Feedback

5.14.1.4.4 Events

Events of the **TOraChangeNotification** class.

For a complete list of the **TOraChangeNotification** class members, see the TOraChangeNotification Members topic.

Published

Name	Description
OnChange	Occurs when data in one of the associated datasets was changed on the server.

See Also

- TOraChangeNotification Class
- TOraChangeNotification Class Members

5.14.1.4.4.1 OnChange Event

Occurs when data in one of the associated datasets was changed on the server.

Class

TOraChangeNotification

Syntax

```
property OnChange: TOraChangeNotificationEvent;
```

Remarks

The OnChange event occurs when data in one of the associated datasets has been changed on the server. To receive change notifications the Enabled property must be set to True. The NotifyType parameter contains the type of the event occurred. The TableChanges parameter contains information on all table changes.

See Also

- Enabled
- TOraChangeNotification Component

5.14.1.5 TOraChangeNotificationOptions Class

Used to specify how query notifications will be generated.

For a list of all members of this type, see TOraChangeNotificationOptions members.

Unit

Ora

Syntax

```
TOraChangeNotificationOptions = class(TPersistent);
```

 Request Support DAC Forum Provide Feedback

5.14.1.5.1 Members

TOraChangeNotificationOptions class overview.

Properties

Name	Description
QueryResultOnly	If the property is set to True, query level granularity is required. Notification should be only generated if the query result set changes.

 Request Support DAC Forum Provide Feedback

5.14.1.5.2 Properties

Properties of the **TOraChangeNotificationOptions** class.

For a complete list of the **TOraChangeNotificationOptions** class members, see the TOraChangeNotificationOptions Members topic.

Published

Name	Description
QueryResultOnly	If the property is set to True, query level granularity is required. Notification should be only generated if the query result set changes.

See Also

- TOraChangeNotificationOptions Class

- TOraChangeNotificationOptions Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.5.2.1 QueryResultOnly Property

If the property is set to True, query level granularity is required. Notification should be only generated if the query result set changes.

Class

TOraChangeNotificationOptions

Syntax

```
property QueryResultOnly: boolean default false;
```

 Request Support DAC Forum Provide Feedback

5.14.1.6 TOraConnectionSSLOptions Class

A class for setting up the SSL options.

For a list of all members of this type, see TOraConnectionSSLOptions members.

Unit

Ora

Syntax

```
TOraConnectionSSLOptions = class(TDAConnectionSSLOptions);
```

Inheritance Hierarchy

TDAConnectionSSLOptions

TOraConnectionSSLOptions

 Request Support DAC Forum Provide Feedback

5.14.1.6.1 Members

TOraConnectionSSLOptions class overview.

Properties

Name	Description
CACert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the certificate authority file.
Cert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the client certificate.
CipherList (inherited from **TDAConnectionSSLOptions**)	Holds the list of allowed SSL ciphers.
Key (inherited from **TDAConnectionSSLOptions**)	Holds the path to the private client key.
ServerCertDN	Used to specify the server's distinguished name (DN).

Request Support DAC Forum Provide Feedback

5.14.1.6.2 Properties

Properties of the **TOraConnectionSSLOptions** class.

For a complete list of the **TOraConnectionSSLOptions** class members, see the TOraConnectionSSLOptions Members topic.

Published

Name	Description
CACert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the certificate authority file.
Cert (inherited from **TDAConnectionSSLOptions**)	Holds the path to the client certificate.
CipherList (inherited from **TDAConnectionSSLOptions**)	Holds the list of allowed SSL ciphers.
Key (inherited from **TDAConnectionSSLOptions**)	Holds the path to the private client key.
ServerCertDN	Used to specify the server's distinguished name (DN).

See Also

- TOraConnectionSSLOptions Class
- TOraConnectionSSLOptions Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.6.2.1 ServerCertDN Property

Used to specify the server's distinguished name (DN).

Class

TOraConnectionSSLOptions

Syntax

```
property ServerCertDN: string;
```

Remarks

Use the ServerCertDN property so specify the server's distinguished name (DN) to enable server DN matching. It checks whether the server is genuine by matching the server's global database name against the DN from the server certificate.

 Request Support DAC Forum Provide Feedback

5.14.1.7 TOraDataSet Class

A class defining the Oracle functionality for a dataset.

For a list of all members of this type, see TOraDataSet members.

Unit

Ora

Syntax

```
TOraDataSet = class(TCustomDADataSet);
```

Remarks

TOraDataSet is a component that defines the Oracle functionality for a dataset. TOraDataSet

can execute queries, fetch rows and control Oracle specific data types. Applications never use TOraDataSet objects directly. Instead they use descendants of TOraDataSet, such as TOraQuery, TSmartQuery, TOraStoredProc, and TOraTable, which inherit its database-related properties and methods.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet

See Also

- TOraQuery
- TSmartQuery
- TOraStoredProc
- TOraTable

5.14.1.7.1 Members

TOraDataSet class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification	Used to receive database change notification messages to refresh dataset when required.
CheckMode	Used to define the check mode before editing a

	record.
CommandTimeout	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption	Used to specify encryption options in a dataset.
FetchAll	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT

	statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL	Indicates whether a SQL statement is a PL/SQL block.
IsQuery	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.

Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking	Used to execute a SQL statement and fetch rows by a separate thread.
Options	Used to specify the behaviour of TOraDataSetObject.
OptionsDS	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.

ReturnParams	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed	Returns the number of rows processed by a query.
SequenceMode	Used to specify the methods used internally to generate a sequenced field.
Session	Used to specify the session in which dataset will be executed.
SmartFetch	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.

SQLType	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is

	fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines whether a parameter with the specified name exists in a dataset.
GetArray	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile	Retrieves a TOraFile object for a field with known name.
GetInterval	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.

GetKeyList	Returns the list of table primary key fields.
GetLob	Retrieves a TOraLob object for a field with known name.
GetLobLocator	Retrieves a TOraLob object for a field with known name.
GetObject	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef	Retrieves a TOraRef object for a field with known name.
GetTable	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext	Opens next cursor or rowset in the statement.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and

	parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.

UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.14.1.7.2 Properties

Properties of the **TOraDataSet** class.

For a complete list of the **TOraDataSet** class members, see the TOraDataSet Members

topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification	Used to receive database change notification messages to refresh dataset when required.
CheckMode	Used to define the check mode before editing a record.
CommandTimeout	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset

	opened after connection is closed.
DMLRefresh	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption	Used to specify encryption options in a dataset.
FetchAll	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL	Indicates whether a SQL statement is a PL/SQL block.
IsQuery	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence	Used to specify the name of

	a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking	Used to execute a SQL statement and fetch rows by a separate thread.
Options	Used to specify the behaviour of TOraDataSetObject.
OptionsDS	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.

ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed	Returns the number of rows processed by a query.
SequenceMode	Used to specify the methods used internally to generate a sequenced field.
Session	Used to specify the session in which dataset will be executed.
SmartFetch	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to

	a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.

UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TOraDataSet Class
- TOraDataSet Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.1 ChangeNotification Property

Used to receive database change notification messages to refresh dataset when required.

Class

TOraDataSet

Syntax

```
property ChangeNotification: TOraChangeNotification;
```

Remarks

Use the ChangeNotification property to associate the component with the TOraChangeNotification component to receive database change notification messages to refresh dataset when required.

See Also

- TOraChangeNotification
- TOraChangeNotification Component
- Options

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.2 CheckMode Property

Used to define the check mode before editing a record.

Class

TOraDataSet

Syntax

```
property CheckMode: TCheckMode default cmNone;
```

Remarks

Use the CheckMode property to define the check mode before editing a record. Checking records is useful in creating concurrent multi-user applications. Set CheckMode to specify what action to take when another user makes modifications to a record. TOraDataSet first refetches record values and compares them with those of a client.

Request Support DAC Forum Provide Feedback

5.14.1.7.2.3 CommandTimeout Property

Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.

Class

TOraDataSet

Syntax

```
property CommandTimeout: integer default 0;
```

Remarks

The wait time is specified in seconds to wait for the command to execute. The default value is 0. The value of 0 indicates there are no time limits (an attempt to execute a command will wait indefinitely).

Request Support DAC Forum Provide Feedback

5.14.1.7.2.4 Cursor Property

Used to fetch data from the cursor parameter and cursor field in Oracle 8.

Class

TOraDataSet

Syntax

```
property Cursor: TOraCursor;
```

Remarks

Use the Cursor property to fetch data from the cursor parameter and cursor field in Oracle 8. You can assign the value of TOraParam.AsCursor or TCursorField.AsCursor to the Cursor property. After assigning you can open the dataset once.

Example

```
OraQuery1.Cursor := OraSQL1.ParamByName('Cur').AsCursor;
OraQuery1.Open;
```

See Also

- TOraCursor
- TOraParam.AsCursor

5.14.1.7.2.5 DMLRefresh Property

Used to refresh record by RETURNING clause when insert or update is performed.

Class

TOraDataSet

Syntax

```
property DMLRefresh: boolean;
```

Remarks

Use the DMLRefresh property to refresh record by RETURNING clause when insert or update is performed. This feature is only for Oracle 8.

The default value is False.

Note: When the DMLRefresh property is set to True, the value of TCustomDADataSet.RefreshOptions is ignored to avoid refetching field values from the server.

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.6 Encryption Property

Used to specify encryption options in a dataset.

Class

TOraDataSet

Syntax

```
property Encryption: TOraEncryption;
```

Remarks

Set the Encryption options for using encryption in a dataset.

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.7 FetchAll Property

Used to request all records of the query from database server when a dataset is being opened.

Class

TOraDataSet

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when a dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

When the FetchAll property is False, the first call to the TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

Request Support DAC Forum Provide Feedback

5.14.1.7.2.8 IsPLSQL Property

Indicates whether a SQL statement is a PL/SQL block.

Class

TOraDataSet

Syntax

```
property IsPLSQL: boolean;
```

Remarks

Use the IsPLSQL property to check whether a SQL statement is a PL/SQL block. TOraDataSet must be prepared beforehand.

IsPLSQL is a read-only property.

See Also

- IsQuery

Request Support DAC Forum Provide Feedback

5.14.1.7.2.9 IsQuery Property

Indicates whether SQL statement returns rows or not.

Class

TOraDataSet

Syntax

```
property IsQuery: boolean;
```

Remarks

When the TOraDataSet component is prepared, it returns True. If SQL statement is SELECT or PL/SQL block it returns the REF CURSOR parameter.

Use the IsQuery property to check whether SQL statement returns rows or not. TOraDataSet returns rows when SQL statement is SELECT or PL/SQL block with the REF CURSOR parameter. TOraDataSet must be prepared beforehand.

IsQuery is a read-only property.

See Also

- IsPLSQL

5.14.1.7.2.10 KeyFields Property

Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.

Class

TOraDataSet

Syntax

```
property KeyFields: string;
```

Remarks

Assign the KeyFields property to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database. To exploit this feature KeyFields may hold a list of semicolon-delimited field names. If KeyFields was not defined before opening the dataset, ROWID pseudo fields are used.

Besides, the KeyFields property may hold the name of a field which will be later assigned with Oracle sequenced values. Beforehand Oracle sequence must be created and its name passed to the KeySequence property.

Sequences are generated when either TDataSet.Insert or TDataSet.Post method is called. Which of these two methods is used to modify the database is determined by the SequenceMode property.

Note: Although keys may be created across a number of table fields, sequence is generated only for the first field found in the KeyFields property.

See Also

- KeySequence
- SequenceMode
- TCustomDADataSet.SQLDelete
- TCustomDADataSet.SQLInsert
- TCustomDADataSet.SQLRefresh
- TCustomDADataSet.SQLUpdate

5.14.1.7.2.11 KeySequence Property

Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Class

`TOraDataSet`

Syntax

```
property KeySequence: string;
```

Remarks

Use the KeySequence property to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.

Note: KeySequence is used by TOraDataSet only if the KeyFields property is assigned.

Example

Here is an example of PL/SQL block generated by TOraDataSet:

```
begin
   SELECT DEPT_SEQ.NEXTVAL
   INTO :DEPTNO
   FROM Dual;
   INSERT INTO DEPT
      (DEPTNO, DNAME)
   VALUES
      (:DEPTNO, :DNAME);
end;
```

See Also

- SequenceMode

Request Support DAC Forum Provide Feedback

5.14.1.7.2.12 LockMode Property

Used to define when to perform the locking of an editing record.

Class

TOraDataSet

Syntax

```
property LockMode: TLockMode;
```

Remarks

Use the LockMode property to define when to perform the locking of an editing record. Locking a record is useful when creating multi-user applications. It prevents the possibility of several

users modifying a record at the same time. Locking is realized through the execution of SELECT FOR UPDATE NOWAIT statement.

Locking is performed by the RefreshRecord method.

To set pessimistic locking use LockMode = lmLockImmediate, CheckMode = cmException. To set optimistic locking use LockMode = lmLockDelayed, CheckMode = cmException.

The default value is lmNone.

See Also

- TCustomDADataSet.Lock
- TCustomDADataSet.SQLLock
- CheckMode

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.13 NonBlocking Property

Used to execute a SQL statement and fetch rows by a separate thread.

Class

TOraDataSet

Syntax

```
property NonBlocking: boolean;
```

Remarks

Set the NonBlocking property to True to execute SQL statement and fetch rows by a separate thread.

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.14 Options Property

Used to specify the behaviour of TOraDataSetObject.

Class

TOraDataSet

Syntax

```
property Options: TOraDataSetOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TOraDataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression property of TField objects by appropriate value.
DeferredLobRead	Used to fetch all Oracle 8 Lob values when they are explicitly requested.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
FullRefresh	Used to refresh fields of all tables by the RefreshRecord method.
PrefetchLobSize	Used to retrieve the LOB length and the LOB data beginning during regular fetch.
PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL	Used to automatically prepare update

	queries before execution.
ProcNamedParams	Used to specify a notation method of passing parameter values to the stored PL/SQL object.
RawAsString	Used to treat all RAW fields as being of string datatype.
ReflectChangeNotify	Used for a dataset component to refresh its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
StatementCache	Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.
TemporaryLobUpdate	Temporary LOBs are used to write input and input/output LOB parameters into database when executing dataset's SQL statements.

See Also

- TCustomDADataSet.Options

5.14.1.7.2.15 OptionsDS Property

Used to specify the behaviour of TOraDataSetObject.

Class

TOraDataSet

Syntax

```
property OptionsDS: TOraDataSetOptionsDS stored False;
```

Remarks

Set the properties of OptionsDS to specify the behaviour of a TOraDataSet object.

Descriptions of all options are in the table below.

Option Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression property of TField objects with the appropriate value.
DeferredLobRead	Used fetch all Oracle 8 Lob values only when they are explicitly requested.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
HideRowId	Used to display the ROWID column.
KeepPrepared	Used to keep TOraDataSet prepared after closing.
RawAsString	Used to treat all RAW fields as being of string datatype.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.

See Also

- TCustomDADataSet.Options

5.14.1.7.2.16 Params Property

Contains the parameters for a query's SQL statement.

Class

TOraDataSet

Syntax

```
property Params: TOraParams stored False;
```

Remarks

The Params parameter contains the parameters for a query's SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

See Also

- TOraParam
- ParamByName

Request Support DAC Forum Provide Feedback

5.14.1.7.2.17 RefreshMode Property

Used to specify when to refresh an editing record.

Class

TOraDataSet

Syntax

```
property RefreshMode: TRefreshMode stored False;
```

Remarks

Use the RefreshMode property to define when to perform a refresh of editing record. Refreshing a record is useful when a table has triggers or fields of a table have default value.

Refresh is performed by the RefreshRecord method.

The default value is rmNone.

Note: RefreshMode is obsolete, and only included for backward compatibility. Use TCustomDADataSet.RefreshOptions instead.

See Also

- TCustomDADataSet.RefreshRecord
- TCustomDADataSet.SQLRefresh

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.18 ReturnParams Property

Used to return a new fields value to dataset after insert or update.

Class

TOraDataSet

Syntax

```
property ReturnParams: boolean stored False;
```

Remarks

Use the ReturnParams property to return the new fields value to dataset after insert or update. The actual value of a field after insert or update may be different from the value stored in local memory if a table has a trigger.

When ReturnParams is True, OUT parameters of SQLInsert and SQLUpdate statements are assigned to the corresponding fields.

OUT parameters can have PL/SQL block or DML statements with the RETURNING clause (for Oracle 8 only).

The default value is False.

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.19 RowsProcessed Property

Returns the number of rows processed by a query.

Class

TOraDataSet

Syntax

```
property RowsProcessed: integer;
```

Remarks

Use the RowsProcessed property to return the number of rows processed by a query. Useful for SELECT, UPDATE and DELETE statements. In case of SELECT statement, RowsProcessed increments by FetchRows.

See Also

- TCustomDADataSet.RowsAffected

Request Support DAC Forum Provide Feedback

5.14.1.7.2.20 SequenceMode Property

Used to specify the methods used internally to generate a sequenced field.

Class

TOraDataSet

Syntax

```
property SequenceMode: TSequenceMode default smPost;
```

Remarks

Set the SequenceMode property to specify which method is used internally to generate a sequenced field.

See Also

- KeyFields
- KeySequence

Request Support DAC Forum Provide Feedback

5.14.1.7.2.21 Session Property

Used to specify the session in which dataset will be executed.

Class

TOraDataSet

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session in which dataset will be executed. If Session is not connected, the Open method calls Session.Connect.

See Also

- TOraSession

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.22 SmartFetch Property

The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.

Class

TOraDataSet

Syntax

```
property SmartFetch: TSmartFetchOptions;
```

See Also

- TSmartFetchOptions

 Request Support DAC Forum Provide Feedback

5.14.1.7.2.23 SQLType Property

Used to get the typecode of the SQL statement being processed by Oracle database server.

Class

TOraDataSet

Syntax

```
property SQLType: integer;
```

Remarks

Read the SQLType property to get the typecode of the SQL statement being processed by Oracle database server.

Request Support DAC Forum Provide Feedback

5.14.1.7.2.24 StrictUpdate Property

Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.

Class

TOraDataSet

Syntax

```
property StrictUpdate: boolean stored False;
```

Remarks

When True, TOraDataSet raises the 'Update failed' exception when the number of updated or deleted records are not equal 1. The exception does not occur when you use a PL/SQL block.

The default value is True.

Note: StrictUpdate is obsolete, and included for backward compatibility only. Use TCustomDADataSet.Options instead.

Request Support DAC Forum Provide Feedback

Reserved.

5.14.1.7.2.25 UpdateObject Property

Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

Class

TOraDataSet

Syntax

```
property UpdateObject: TOraUpdateSQL;
```

Remarks

The UpdateObject property specifies an update object component which provides SQL statements that perform updates of the read-only datasets when cached updates are enabled.

Request Support DAC Forum Provide Feedback

5.14.1.7.3 Methods

Methods of the **TOraDataSet** class.

For a complete list of the **TOraDataSet** class members, see the TOraDataSet Members topic.

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.

BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.

FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam	Determines whether a parameter with the specified name exists in a dataset.
GetArray	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile	Retrieves a TOraFile object for a field with known name.
GetInterval	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from	Provides a list of available key field names.

TCustomDADataSet)	
GetKeyList	Returns the list of table primary key fields.
GetLob	Retrieves a TOraLob object for a field with known name.
GetLobLocator	Retrieves a TOraLob object for a field with known name.
GetObject	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef	Retrieves a TOraRef object for a field with known name.
GetTable	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext	Opens next cursor or rowset in the statement.
ParamByName	Sets or uses parameter information for a specific parameter based on its name.

Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.

UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TOraDataSet Class
- TOraDataSet Class Members

Request Support DAC Forum Provide Feedback

5.14.1.7.3.1 CreateProcCall Method

Generates the stored procedure call.

Class

TOraDataSet

Syntax

```
procedure CreateProcCall(Name: string; Overload: integer = 0);
```

Parameters

Name

Holds the name of the stored procedure.

Overload

Holds the numder of the overloaded procedure.

Remarks

Call the CreateProcCall method to assign a PL/SQL block that calls stored procedure specified by Name to the SQL property. Overload parameter must contain the number of the overloaded procedure. Retrieves the information about the procedure parameters from Oracle. After calling CreateProcCall you can execute stored procedure by the Execute

method.

See Also

- TCustomDADataSet.Execute
- TCustomDAConnection.ExecProc
- TOraStoredProc

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.2 ErrorOffset Method

Returns the parse error offset.

Class

TOraDataSet

Syntax

```
function ErrorOffset: integer;
```

Return Value

the parse error offset.

Remarks

Call the ErrorOffset method to return the parse error offset for a SQL statement. Check ErrorOffset after TOraDataSet raises an exception.

See Also

- GetErrorPos

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.3 FindParam Method

Determines whether a parameter with the specified name exists in a dataset.

Class

TOraDataSet

Syntax

```
function FindParam(const Value: string): TOraParam;
```

Parameters

Value

Holds the name of the param to search for.

Return Value

a TOraParam object for the specified Name.

Remarks

Call the FindParam method to determine if parameter with the specified name exists in a dataset. Name is the name of the param for which to search. If FindParam finds a param with a matching name, it returns a TOraParam object for the specified Name. Otherwise it returns nil.

See Also

- Params
- ParamByName

5.14.1.7.3.4 GetArray Method

Retrieves a TOraArray object for a field when only its name is known.

Class

TOraDataSet

Syntax

```
function GetArray(const FieldName: string): TOraArray;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraArray object for a field with known name.

Remarks

Call the GetArray method to retrieve a TOraArray object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftArray type.

See Also

- TOraArray
- TCustomDADataSet.GetDataType
- TOraParam.AsArray

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.5 GetErrorPos Method

Returns a row and column of parse error for a SQL statement.

Class

TOraDataSet

Syntax

```
procedure GetErrorPos(var Row: integer; var Col: integer);
```

Parameters

Row

Holds the row number.

Col

Holds the column number.

Remarks

Call the GetErrorPos method to return a row and column of parse error for a SQL statement. Use GetErrorPos after TOraDataSet raises an exception.

See Also

- ErrorOffset

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.6 GetFile Method

Retrieves a TOraFile object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetFile(const FieldName: string): TOraFile;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraFile object for a field with known name.

Remarks

Call the GetFile method to retrieve a TOraFile object for a field when only its name is known. FieldName is the name of an existing field. The field should have ftBFile.

See Also

- TOraFile
- TCustomDADataSet.GetDataType
- TBFileField
- TOraParam.AsBFile

5.14.1.7.3.7 GetInterval Method

Retrieves a TOraInterval object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetInterval(const FieldName: string): TOraInterval;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraInterval object for a field with known name.

Remarks

Call the GetInterval method to retrieve a TOraInterval object for a field when only its name is known. FieldName is the name of an existing field. The field should have ftIntervalYM or ftIntervalDS.

See Also

- TOraInterval
- TOraParam.AsInterval

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.8 GetKeyList Method

Returns the list of table primary key fields.

Class

TOraDataSet

Syntax

```
function GetKeyList(TableName: string; List: TStrings): string;
```

Parameters

TableName

Holds the table name.

List

Return Value

the list of table primary key fields.

Remarks

Call the GetKeyList method to get the list of table primary key fields.

Request Support DAC Forum Provide Feedback

5.14.1.7.3.9 GetLob Method

Retrieves a TOraLob object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetLob(const FieldName: string): TOraLob;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraLob object for a field with known name.

Remarks

Call the GetLob method to retrieve a TOraLob object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftOraClob or ftOraBlob type.

See Also

- TOraLob
- TCustomDADataSet.GetDataType
- TOraParam.AsBLOBLocator

Request Support DAC Forum Provide Feedback

5.14.1.7.3.10 GetLobLocator Method

Retrieves a TOraLob object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetLobLocator(const FieldName: string): TOraLob;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraLob object for a field with known name.

Remarks

Call the GetLobLocator method to retrieve a TOraLob object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftOraClob or ftOraBlob type.

Note: GetLobLocator is an obsolete method. In newer projects call GetLob instead.

See Also

- GetLob
- TOraLob
- TCustomDADataSet.GetDataType
- TOraParam.AsBLOBLocator

5.14.1.7.3.11 GetObject Method

Retrieves a TOraObject object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetObject(const FieldName: string): TOraObject;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraObject object for a field with known name.

Remarks

Call the GetObject method to retrieve a TOraObject object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftObject, ftReference, ftArray or ftTable type.

See Also

- TOraObject
- TCustomDADataSet.GetDataType
- TOraParam.AsObject

Request Support DAC Forum Provide Feedback

5.14.1.7.3.12 GetRef Method

Retrieves a TOraRef object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetRef(const FieldName: string): TOraRef;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraRef object for a field with known name.

Remarks

Call the GetRef method to retrieve a TOraRef object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftReference type.

See Also

- TOraRef
- TCustomDADataSet.GetDataType
- TOraNestedTable.Ref
- TOraParam.AsRef

Request Support DAC Forum Provide Feedback

5.14.1.7.3.13 GetTable Method

Retrieve a TOraNestTable object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetTable(const FieldName: string): TOraNestTable;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraNestTable object for a field with known name.

Remarks

Call the GetTable to retrieve a TOraNestTable object for a field when only its name is known. FieldName is the name of an existing field. The field should have the ftTable type.

See Also

- TOraNestTable

- TCustomDADataSet.GetDataType
- TOraNestedTable.Table
- TOraParam.AsTable

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.14 GetTimeStamp Method

Retrieves a TOraTimeStamp object for a field with known name.

Class

TOraDataSet

Syntax

```
function GetTimeStamp(const FieldName: string): TOraTimeStamp;
```

Parameters

FieldName

Holds the name of an existing field.

Return Value

a TOraTimeStamp object for a field with known name.

Remarks

Call the GetTimeStamp method to retrieve a TOraTimeStamp object for a field when only its name is known. FieldName is the name of an existing field. The field should have ftTimeStamp.

See Also

- TOraTimeStamp
- TOraParam.AsTimeStamp

 Request Support DAC Forum Provide Feedback

5.14.1.7.3.15 OpenNext Method

Opens next cursor or rowset in the statement.

Class

TOraDataSet

Syntax

```
function OpenNext: boolean;
```

Return Value

True, if DataSet opens.

Remarks

Call the OpenNext method to get the second and other Cursors or ResultSets while executing a query. If DataSet opens, it returns True. If there are no cursors or record sets to be represented, it will return False, and the current record set will be closed.

Example for working with cursors:

```
 OraQuery1.SQL.Text := 'BEGIN ' +
                       '  OPEN :Cur1 FOR SELECT * FROM Dept; ' +
                       '  OPEN :Cur2 FOR SELECT * FROM Emp; ' +
                       'END;';
  OraQuery1.Params[0].DataType := ftCursor;
  OraQuery1.Params[0].ParamType := ptOutput;
  OraQuery1.Params[1].DataType := ftCursor;
  OraQuery1.Params[1].ParamType := ptOutput;
  OraQuery1.Open;     // open first cursor
  OraQuery1.OpenNext; // open next cursor
```

Example for working with Implicit Result Sets in Oracle 12c and higher:

```
  OraQuery1.SQL.Text := 'DECLARE ' +
                       '  dept_cur SYS_REFCURSOR; ' +
                       '  emp_cur SYS_REFCURSOR; ' +
                       'BEGIN ' +
                       '  OPEN dept_cur FOR SELECT * FROM Dept; ' +
                       '  DBMS_SQL.RETURN_RESULT(dept_cur); ' +
                       '  OPEN emp_cur FOR SELECT * FROM Emp; ' +
                       '  DBMS_SQL.RETURN_RESULT(emp_cur); ' +
                       'END;';
  OraQuery1.Open;     // open first result set
  OraQuery1.OpenNext; // open next result set
```

5.14.1.7.3.16 ParamByName Method

Sets or uses parameter information for a specific parameter based on its name.

Class

TOraDataSet

Syntax

```
function ParamByName(const Value: string): TOraParam;
```

Parameters

Value

holds the name of the parameter to retrieve information for.

Return Value

a object.

Remarks

Call the ParamByName method to set or use parameter information for a specific parameter based on its name. Name is the name of the parameter for which to retrieve information. ParamByName is used to set a parameter's value at runtime and returns a TOraParam object.

Example

The following statement retrieves the current value of a parameter called "Contact" into an edit box:

```
Edit1.Text := Query1.ParamsByName('Contact').AsString;
```

See Also

- TOraParam
- Params
- TCustomDADataSet.FindParam

5.14.1.8 TOraDataSetField Class

A class providing access to Oracle nested datasets.

For a list of all members of this type, see TOraDataSetField members.

Unit

Ora

Syntax

```
TOraDataSetField = class(TDataSetField);
```

Remarks

TOraDataSetField provides access to Oracle nested datasets.

 Request Support DAC Forum Provide Feedback

5.14.1.8.1 Members

TOraDataSetField class overview.

Properties

Name	Description
Modified	Indicates whether tafield was modified.

 Request Support DAC Forum Provide Feedback

5.14.1.8.2 Properties

Properties of the **TOraDataSetField** class.

For a complete list of the **TOraDataSetField** class members, see the TOraDataSetField Members topic.

Public

Name	Description

Modified	Indicates whether tafield was modified.

See Also

- TOraDataSetField Class
- TOraDataSetField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.8.2.1 Modified Property

Indicates whether tafield was modified.

Class

TOraDataSetField

Syntax

```
property Modified: boolean;
```

Remarks

The Modified proprty indicates whether a field was modified. The property is writable.

 Request Support DAC Forum Provide Feedback

5.14.1.9 TOraDataSetOptions Class

This class allows setting up the behaviour of the TOraDataSet class.

For a list of all members of this type, see TOraDataSetOptions members.

Unit

Ora

Syntax

```
TOraDataSetOptions = class(TOraDataSetOptionsDS);
```

Remarks

Cached Gets a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session. If you execute many different SELECT statements this option may significantly increase the performance of your application. But using this property you may easily step over maximum open cursors on Oracle server. And thus you must be attentive when using the Cached option. This option is only available with Oracle 9.2i and higher. It will work only if TOraSession.Options is set to True.

Inheritance Hierarchy

TDADataSetOptions

TOraDataSetOptionsDS

TOraDataSetOptions

Request Support DAC Forum Provide Feedback

5.14.1.9.1 Members

TOraDataSetOptions class overview.

Properties

Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression

	property of TField objects by appropriate value.
DeferredLobRead	Used to fetch all Oracle 8 Lob values when they are explicitly requested.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
FullRefresh	Used to refresh fields of all tables by the RefreshRecord method.
HideRowId (inherited from **TOraDataSetOptionsDS**)	Used to display the ROWID column.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement

KeepPrepared (inherited from **TOraDataSetOptionsDS**)	Used to keep TOraDataSet prepared after closing.
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
PrefetchLobSize	Used to retrieve the LOB length and the LOB data beginning during regular fetch.
PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
ProcNamedParams	Used to specify a notation method of passing parameter values to the stored PL/SQL object.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of

	records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RawAsString	Used to treat all RAW fields as being of string datatype.
ReflectChangeNotify	Used for a dataset component to refresh its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StatementCache	Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to

	raise an exception when the number of updated or deleted records is not equal 1.
TemporaryLobUpdate	Temporary LOBs are used to write input and input/ output LOB parameters into database when executing dataset's SQL statements.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support DAC Forum Provide Feedback

5.14.1.9.2 Properties

Properties of the **TOraDataSetOptions** class.

For a complete list of the **TOraDataSetOptions** class members, see the TOraDataSetOptions Members topic.

Public

Name	Description
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from	Used to store values of the BLOB fields in compressed

TDADataSetOptions)	form.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to

	perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of

	commands that can be executed in a batch.

Published

Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression property of TField objects by appropriate value.
DeferredLobRead	Used to fetch all Oracle 8 Lob values when they are explicitly requested.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo	Used to perform an additional query to get information about returned fields and the tables they belong to.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
FullRefresh	Used to refresh fields of all tables by the RefreshRecord method.
HideRowId (inherited from **TOraDataSetOptionsDS**)	Used to display the ROWID column.
KeepPrepared (inherited from **TOraDataSetOptionsDS**)	Used to keep TOraDataSet prepared after closing.
PrefetchLobSize	Used to retrieve the LOB length and the LOB data beginning during regular fetch.

PrefetchRows	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL	Used to automatically prepare update queries before execution.
ProcNamedParams	Used to specify a notation method of passing parameter values to the stored PL/SQL object.
RawAsString	Used to treat all RAW fields as being of string datatype.
ReflectChangeNotify	Used for a dataset component to refresh its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
StatementCache	Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.
TemporaryLobUpdate	Temporary LOBs are used to write input and input/ output LOB parameters into database when executing dataset's SQL statements.

See Also

- TOraDataSetOptions Class
- TOraDataSetOptions Class Members

5.14.1.9.2.1 AutoClose Property

Used to close OCI cursor after fetching all rows.

Class

TOraDataSetOptions

Syntax

```
property AutoClose: boolean stored True;
```

Remarks

Use the AutoClose property for TOraDataSet to close OCI cursor after fetching all rows. Allows reducing the number of opened cursors on the server.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.2 CacheLobs Property

Used to allocate local memory buffer to hold a copy of the Lob content.

Class

TOraDataSetOptions

Syntax

```
property CacheLobs: boolean stored True;
```

Remarks

If True, (the default value) then local memory buffer is allocated to hold a copy of the Lob content. If this option is set to False, it is highly recommended to set the DeferredLobRead option to True. Otherwise, LOB values are fetched to the dataset, and it can result in performance loss.

Note: The CacheLobs option controls the way Lob objects are handled while an application fetches records from the database. Setting CacheLobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since Lob objects are only transferred on demand; less memory is needed on the client side because returned

record sets do not hold contents of the Lob fields. Actual value for the Lob field is passed to the client only when a data-aware control requests it.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.3 DefaultValues Property

Used for TOraDataSet to fill the DefaultExpression property of TField objects by appropriate value.

Class

TOraDataSetOptions

Syntax

```
property DefaultValues: boolean stored True;
```

Remarks

If True, TOraDataSet fills the DefaultExpression property of TField objects by appropriate value.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.4 DeferredLobRead Property

Used to fetch all Oracle 8 Lob values when they are explicitly requested.

Class

TOraDataSetOptions

Syntax

```
property DeferredLobRead: boolean stored True;
```

Remarks

If True, all Oracle 8 Lob values are only fetched when they are explicitly requested. Otherwise entire record set with any Lob values is returned when dataset is opened. Whether Lob

values are cached locally to be reused later or not is controlled by CacheLobs option.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.5 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

TOraDataSetOptions

Syntax

```
property EnableBCD: boolean;
```


Request Support DAC Forum Provide Feedback

5.14.1.9.2.6 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

TOraDataSetOptions

Syntax

```
property EnableFMTBCD: boolean;
```


Request Support DAC Forum Provide Feedback

5.14.1.9.2.7 ExtendedFieldsInfo Property

Used to perform an additional query to get information about returned fields and the tables they belong to.

Class

TOraDataSetOptions

Syntax

```
property ExtendedFieldsInfo: boolean;
```

Remarks

If True, an additional query is performed to get information about returned fields and the tables they belong to. True by default for TSmartQuery, and False by default for other TOraDataSet descendants.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.8 FieldsAsString Property

Used to treat all non-BLOB fields as being of string datatype.

Class

TOraDataSetOptions

Syntax

```
property FieldsAsString: boolean stored True;
```

Remarks

If True, all non-BLOB fields are treated as being of string datatype.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.9 FullRefresh Property

Used to refresh fields of all tables by the RefreshRecord method.

Class

TOraDataSetOptions

Syntax

```
property FullRefresh: boolean;
```

Remarks

Set the FullRefresh property to True to refresh fields of all tables by the RefreshRecord method. To perform full refreshing TCustomSmartQuery executes modified SELECT statement defined by the SQL property. When FullRefresh is False TCustomSmartQuery performs refreshing fields of the updating table only. The default value is False.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.10 PrefetchLobSize Property

Used to retrieve the LOB length and the LOB data beginning during regular fetch.

Class

TOraDataSetOptions

Syntax

```
property PrefetchLobSize: Integer default 0;
```

Remarks

Use the PrefetchLobSize option to retrieve the LOB length and the chunk size as well as the beginning of the LOB data along with the locator during regular fetch. The PrefetchLobSize property specifies the size of LOB data that will be prefetched. If the total LOB size is less or equals to PrefetchLobSize, then all LOB data will be fetched during regular fetch without additional round trips that can improve performance greatly.

Note: Prefetching LOB data is available in Oracle 11 and higher.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.11 PrefetchRows Property

Used to set the number of rows to be prefetched during the execution of a query.

Class

TOraDataSetOptions

Syntax

```
property PrefetchRows: integer default 0;
```

Remarks

Use the PrefetchRows property to set the number of rows to be prefetched during the execution of a query. Setting the property to a value greater than 0 reduces the server round-trip count, which increases the performance of the application. The default value is 0 - the number of prefetched rows is determined automatically. To disable row prefetching, set the property to -1.

Note: Some queries (for example, query SELECT * FROM DUAL CONNECT BY LEVEL <= 5 returns 1 row when prefetching is enabled, and 5 rows when it is disabled) can return invalid rows count when prefetching is enabled.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.12 PrepareUpdateSQL Property

Used to automatically prepare update queries before execution.

Class

TOraDataSetOptions

Syntax

```
property PrepareUpdateSQL: boolean;
```

Remarks

If True, update queries are automatically prepared before executing.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.13 ProcNamedParams Property

Used to specify a notation method of passing parameter values to the stored PL/SQL object.

Class

TOraDataSetOptions

Syntax

```
property ProcNamedParams: boolean default False;
```

Remarks

Positional Notation is used if OraStoredProc.Options.ProcNamedParams = False (default value). If True, Named Notation is used.

Named Notation allows passing parameter values in any order regardless of the position.

Example

Sample of stored proc call with Named Notation:

```
credit_acct(amount => amt, acct_no => acct);
```


Request Support DAC Forum Provide Feedback

5.14.1.9.2.14 Raw AsString Property

Used to treat all RAW fields as being of string datatype.

Class

TOraDataSetOptions

Syntax

```
property RawAsString: boolean stored True;
```

Remarks

If True, all RAW fields are treated as being of string datatype, i.e. represented as hexadecimal string.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.15 ReflectChangeNotify Property

Used for a dataset component to refresh its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.

Class

TOraDataSetOptions

Syntax

```
property ReflectChangeNotify: boolean default False;
```

Remarks

If True and the TOraDataSet.ChangeNotification property is set, dataset component refreshes its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.

Request Support DAC Forum Provide Feedback

5.14.1.9.2.16 ScrollableCursor Property

Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.

Class

TOraDataSetOptions

Syntax

```
property ScrollableCursor: boolean stored True;
```

Remarks

If True, TOraDataSet does not cache data on the client side but uses scrollable server cursor (available since Oracle 9 only). This option can be used to reduce memory usage, because dataset stores only current fetched block. Unlike the TCustomDADataSet.UniDirectional option ScrollableCursor allows bidirectional dataset navigation. Note that scrollable cursor is read-only by its nature.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.17 StatementCache Property

Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.

Class

TOraDataSetOptions

Syntax

```
property StatementCache: boolean default False;
```

Remarks

OCI statement cache is enabled when you set TOraSessionOptions.StatementCacheSize in TOraSession.Options to a positive value. Set TOraSessionOptions.StatementCacheSize to 0 (default) or TOraSession.Options.StatemantCache to false if you don't want the statements to be cached.

 Request Support DAC Forum Provide Feedback

5.14.1.9.2.18 TemporaryLobUpdate Property

Temporary LOBs are used to write input and input/output LOB parameters into database when executing dataset's SQL statements.

Class

TOraDataSetOptions

Syntax

```
property TemporaryLobUpdate: boolean default False;
```

Remarks

Set the TemporaryLobUpdate property to True to use temporary LOBs to write input and input/output LOB parameters into database when executing dataset's SQL statements.

Note: CacheLobs option controls the way Lob objects are handled while the application fetches records from the database. Setting CacheLobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since Lob objects are only transferred on demand; less memory is needed on the client side because returned record sets do not hold contents of Lob fields. Actual value for the Lob field is passed to the client only when a data-aware control requests it.

 Request Support DAC Forum Provide Feedback

5.14.1.10 TOraDataSetOptionsDS Class

This class allows setting up the behaviour of the TOraDaatSet class (this property is obsolete).

For a list of all members of this type, see TOraDataSetOptionsDS members.

Unit

Ora

Syntax

```
TOraDataSetOptionsDS = class(TDADataSetOptions);
```

Remarks

Note: The OptionsDS property is obsolete. It is provided for backward compatibility only.

Inheritance Hierarchy

TDADataSetOptions

TOraDataSetOptionsDS

 Request Support DAC Forum Provide Feedback

5.14.1.10.1 Members

TOraDataSetOptionsDS class overview.

Properties

Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression property of TField objects with the appropriate value.
DeferredLobRead	Used fetch all Oracle 8 Lob values only when they are explicitly requested.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
HideRowId	Used to display the ROWID column.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the

	generated INSERT statement
KeepPrepared	Used to keep TOraDataSet prepared after closing.
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RawAsString	Used to treat all RAW fields as being of string datatype.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.

RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support DAC Forum Provide Feedback

5.14.1.10.2 Properties

Properties of the **TOraDataSetOptionsDS** class.

For a complete list of the **TOraDataSetOptionsDS** class members, see the TOraDataSetOptionsDS Members topic.

Public

Name	Description
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CompressBlobMode (inherited from **TDADataSetOptions**)	Used to store values of the BLOB fields in compressed form.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.

LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or

	can not be updated.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Published

Name	Description
AutoClose	Used to close OCI cursor after fetching all rows.
CacheLobs	Used to allocate local memory buffer to hold a copy of the Lob content.
DefaultValues	Used for TOraDataSet to fill the DefaultExpression property of TField objects with the appropriate value.
DeferredLobRead	Used fetch all Oracle 8 Lob values only when they are explicitly requested.
FieldsAsString	Used to treat all non-BLOB fields as being of string datatype.
HideRowId	Used to display the ROWID column.
KeepPrepared	Used to keep TOraDataSet prepared after closing.

RawAsString	Used to treat all RAW fields as being of string datatype.
ScrollableCursor	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.

See Also

- TOraDataSetOptionsDS Class
- TOraDataSetOptionsDS Class Members

Request Support DAC Forum Provide Feedback

5.14.1.10.2.1 AutoClose Property

Used to close OCI cursor after fetching all rows.

Class

TOraDataSetOptionsDS

Syntax

```
property AutoClose: boolean stored False default False;
```

Remarks

Use the AutoClose property for TOraDataSet to close OCI cursor after fetching all rows.

Allows to reduce the number of opened cursors on the server.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.2 CacheLobs Property

Used to allocate local memory buffer to hold a copy of the Lob content.

Class

TOraDataSetOptionsDS

Syntax

```
property CacheLobs: boolean stored False default True;
```

Remarks

If True, (the default value) then local memory buffer is allocated to hold a copy of the Lob content. See the notes below for further details.

Note: CacheLobs option controls the way Lob objects are handled while the application fetches records from the database. Setting CacheLobs to False may bring up the following benefits for time-critical applications: reduced traffic over the network since Lob objects are only transferred on demand; less memory is needed on the client side because returned record sets do not hold contents of Lob fields. Actual value for the Lob field is passed to the client only when a data-aware control requests it.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.3 DefaultValues Property

Used for TOraDataSet to fill the DefaultExpression property of TField objects with the appropriate value.

Class

TOraDataSetOptionsDS

Syntax

```
property DefaultValues: boolean stored False;
```

Remarks

If True, TOraDataSet fills the DefaultExpression property of TField objects with the appropriate value.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.4 DeferredLobRead Property

Used fetch all Oracle 8 Lob values only when they are explicitly requested.

Class

TOraDataSetOptionsDS

Syntax

```
property DeferredLobRead: boolean stored False default False;
```

Remarks

If True, all Oracle 8 Lob values are only fetched when they are explicitly requested. Otherwise entire record set with any Lob values is returned when dataset is opened. Whether Lob values are cached locally to be reused later or not is controlled by CacheLobs option.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.5 FieldsAsString Property

Used to treat all non-BLOB fields as being of string datatype.

Class

TOraDataSetOptionsDS

Syntax

```
property FieldsAsString: boolean stored False default False;
```

Remarks

If True, all non-BLOB fields are treated as being of string datatype.

Request Support DAC Forum Provide Feedback

5.14.1.10.2.6 HideRow Id Property

Used to display the ROWID column.

Class

TOraDataSetOptionsDS

Syntax

```
property HideRowId: boolean default True;
```

Remarks

Used to display the RowId service field. By default, the Visible property for this field is set to False.

5.14.1.10.2.7 KeepPrepared Property

Used to keep TOraDataSet prepared after closing.

Class

TOraDataSetOptionsDS

Syntax

```
property KeepPrepared: boolean stored False;
```

Remarks

If True, TOraDataSet remains prepared after closing. It allows to avoid needless reopening cursor on the server.

5.14.1.10.2.8 Raw AsString Property

Used to treat all RAW fields as being of string datatype.

Class

TOraDataSetOptionsDS

Syntax

```
property RawAsString: boolean stored False default False;
```

Remarks

If True, all RAW fields are treated as being of string datatype, i.e. represented as hexadecimal string.

5.14.1.10.2.9 ScrollableCursor Property

Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.

Class

TOraDataSetOptionsDS

Syntax

```
property ScrollableCursor: boolean stored False default False;
```

Remarks

If True, TOraDataSet does not cache data on the client side but uses scrollable server cursor (available since Oracle 9 only). This option can be used to reduce memory usage, because dataset stores only current fetched block. Unlike the TCustomDADataSet.UniDirectional option ScrollableCursor allows bidirectional dataset navigation. Note that scrollable cursor is read-only by its nature.

5.14.1.11 TOraDataSource Class

TOraDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.

For a list of all members of this type, see TOraDataSource members.

Unit

Ora

Syntax

```
TOraDataSource = class(TCRDataSource);
```

Remarks

TOraDataSource provides an interface between an ODAC dataset components and data-aware controls on a form.

TOraDataSource inherits its functionality directly from the TDataSource component.

At design-time assign individual data-aware components' DataSource properties from their drop-down listboxes.

Inheritance Hierarchy

TCRDataSource

TOraDataSource

5.14.1.11.1 Members

TOraDataSource class overview.

5.14.1.12 TOraEncryptor Class

The class that performs encrypting and decrypting of data.

For a list of all members of this type, see TOraEncryptor members.

Unit

Ora

Syntax

```
TOraEncryptor = class(TCREncryptor);
```

Inheritance Hierarchy

TCREncryptor

TOraEncryptor

5.14.1.12.1 Members

TOraEncryptor class overview.

Properties

Name	Description
DataHeader (inherited from **TCREncryptor**)	Specifies whether the additional information is stored with the encrypted data.
EncryptionAlgorithm (inherited from **TCREncryptor**)	Specifies the algorithm of data encryption.
HashAlgorithm (inherited from **TCREncryptor**)	Specifies the algorithm of generating hash data.
InvalidHashAction (inherited from **TCREncryptor**)	Specifies the action to perform on data fetching when hash data is invalid.
Password (inherited from **TCREncryptor**)	Used to set a password that is used to generate a key for encryption.

Methods

Name	Description

SetKey (inherited from **TCREncryptor**)	Sets a key, using which data is encrypted.

 Request Support DAC Forum Provide Feedback

5.14.1.13 TOraIntervalField Class

A class providing access to the Oracle interval fields.

For a list of all members of this type, see TOraIntervalField members.

Unit

Ora

Syntax

```
TOraIntervalField = class(TField);
```

Remarks

TOraIntervalField provides access to the Oracle interval fields. Unlike other TField descendants the TOraIntervalField.DataType property has two valid values ftIntervalYM and ftIntervalDS depending on the type of the interval.

You can access actual interval value using properties AsString and AsOraInterval properties.

See Also

- TOraInterval

 Request Support DAC Forum Provide Feedback

5.14.1.13.1 Members

TOraIntervalField class overview.

Properties

Name	Description
AsInterval	Used to provide access to a

	TOraInterval object.
FracPrecision	Used to get or set the number of digits used to represent fractional seconds when getting interval value as string.
LeadPrecision	Used to get or set the number of digits that are used to represent the leading interval part when getting the interval value as string.

Request Support DAC Forum Provide Feedback

5.14.1.13.2 Properties

Properties of the **TOraIntervalField** class.

For a complete list of the **TOraIntervalField** class members, see the TOraIntervalField Members topic.

Public

Name	Description
AsInterval	Used to provide access to a TOraInterval object.

Published

Name	Description
FracPrecision	Used to get or set the number of digits used to represent fractional seconds when getting interval value as string.
LeadPrecision	Used to get or set the number of digits that are used to represent the leading interval part when getting the interval value as string.

See Also

- TOraIntervalField Class
- TOraIntervalField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.13.2.1 AsInterval Property

Used to provide access to a TOraInterval object.

Class

TOraIntervalField

Syntax

```
property AsInterval: TOraInterval;
```

Remarks

Use the AsInterval property to provide access to a TOraInterval object you can use for manipulations with the interval value.

See Also

- TOraInterval

 Request Support DAC Forum Provide Feedback

5.14.1.13.2.2 FracPrecision Property

Used to get or set the number of digits used to represent fractional seconds when getting interval value as string.

Class

TOraIntervalField

Syntax

```
property FracPrecision: integer default 6;
```

Remarks

Use the FracPrecision property to get or set the number of digits used to represent fractional seconds when getting interval value as string. This property affects only INTERVAL DAY TO SECOND (ftIntervalDS). The default value of the property is 6.

See Also

- TOraInterval.FracPrecision
- TOraInterval.AsString

Request Support DAC Forum Provide Feedback

5.14.1.13.2.3 LeadPrecision Property

Used to get or set the number of digits that are used to represent the leading interval part when getting the interval value as string.

Class

TOraIntervalField

Syntax

```
property LeadPrecision: integer default 2;
```

Remarks

Use the LeadPrecision property to get or set the number of digits that are used to represent the leading interval part when getting the interval value as string. The default value of the property is 2.

See Also

- TOraInterval.LeadPrecision
- TOraInterval.AsString

Request Support DAC Forum Provide Feedback

5.14.1.14 TOraMetaData Class

A component for obtaining metainformation about database objects from the server.

For a list of all members of this type, see TOraMetaData members.

Unit

Ora

Syntax

```
TOraMetaData = class(TDAMetaData);
```

Remarks

The TOraMetaData component is used to obtain metainformation from the server about objects in the database, such as tables, table columns, stored procedures, etc.

Inheritance Hierarchy

TMemDataSet
  TDAMetaData
    TOraMetaData

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug
- DBMonitor

5.14.1.14.1 Members

TOraMetaData class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates

	for a dataset.
Connection (inherited from **TDAMetaData**)	Used to specify a connection object to use to connect to a data store.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MetaDataKind (inherited from **TDAMetaData**)	Used to specify which kind of metainformation to show.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Restrictions (inherited from **TDAMetaData**)	Used to provide one or more conditions restricting the list of objects to be described.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetMetaDataKinds (inherited from **TDAMetaData**)	Used to get values acceptable in the MetaDataKind property.
GetRestrictions (inherited from **TDAMetaData**)	Used to find out which restrictions are applicable to a certain MetaDataKind.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to

	the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not

	handle the updates.

Request Support DAC Forum Provide Feedback

5.14.1.15 TOraNestedTable Class

A component for controlling nested table data.

For a list of all members of this type, see TOraNestedTable members.

Unit

Ora

Syntax

```
TOraNestedTable = class(TMemDataSet);
```

Remarks

Nested table is a dataset component that encapsulates a database table that is nested as a field within another table. Use TOraNestedTable to access data contained in a nested dataset. A nested table provides much of the functionality of a table component, with the difference that the data it accesses is stored in a nested table.

TOraNestedTable is derived from the TMemDataSet component.

Inheritance Hierarchy

TMemDataSet
  TOraNestedTable

See Also

- TOraNestTable
- TOraRef

Request Support DAC Forum Provide Feedback

5.14.1.15.1 Members

TOraNestedTable class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Ref	Used to assign reference data to TOraNestedTable.
Table	Used to assign nested table data to TOraNestedTable.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description

ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when

	cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

5.14.1.15.2 Properties

Properties of the **TOraNestedTable** class.

For a complete list of the **TOraNestedTable** class members, see the TOraNestedTable Members topic.

Public

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
Ref	Used to assign reference data to TOraNestedTable.
Table	Used to assign nested table data to TOraNestedTable.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.

UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TOraNestedTable Class
- TOraNestedTable Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.15.2.1 Ref Property

Used to assign reference data to TOraNestedTable.

Class

TOraNestedTable

Syntax

```
property Ref: TOraRef;
```

Remarks

Use the Ref property to assign reference data to TOraNestedTable. After assigning you can call the Open method to browse the reference data.

See Also

- TOraRef
- TOraParam.AsRef
- TOraDataSet.GetRef

 Request Support DAC Forum Provide Feedback

5.14.1.15.2.2 Table Property

Used to assign nested table data to TOraNestedTable.

Class

TOraNestedTable

Syntax

```
property Table: TOraNestTable;
```

Remarks

Use the Table property to assign nested table data to TOraNestedTable. After assigning you can call the Open method to browse the nested table data.

Example

```
OraNestedTable1.Table := OraSQL1.ParamByName('Content').AsTable;
OraNestedTable1.Open;
```

See Also

- TOraNestTable
- TOraParam.AsTable
- TOraDataSet.GetTable

 Request Support DAC Forum Provide Feedback

5.14.1.16 TOraNumberField Class

A class providing access to the Oracle number fields.

For a list of all members of this type, see TOraNumberField members.

Unit

Ora

Syntax

```
TOraNumberField = class(TNumericField);
```

Remarks

TOraNumberField provides access to Oracle number fields. The TOraNumberField.DataType property values equals to ftNumber.

You can access actual number value using AsString, AsInteger and AsFloat properties.

See Also

- TOraNumber

 Request Support DAC Forum Provide Feedback

5.14.1.16.1 Members

TOraNumberField class overview.

Properties

Name	Description
AsNumber	Used to provide access to a TOraNumber object.

 Request Support DAC Forum Provide Feedback

5.14.1.16.2 Properties

Properties of the **TOraNumberField** class.

For a complete list of the **TOraNumberField** class members, see the TOraNumberField Members topic.

Public

Name	Description
AsNumber	Used to provide access to a TOraNumber object.

See Also

- TOraNumberField Class
- TOraNumberField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.16.2.1 AsNumber Property

Used to provide access to a TOraNumber object.

Class

TOraNumberField

Syntax

```
property AsNumber: TOraNumber;
```

Remarks

Use the AsNumber property to provide access to a TOraNumber object that you can use for manipulations with the number value.

See Also

- TOraNumber

Request Support DAC Forum Provide Feedback

5.14.1.17 TOraParam Class

A class that is used to set the values of individual parameters passed with queries or stored procedures.

For a list of all members of this type, see TOraParam members.

Unit

Ora

Syntax

```
TOraParam = class(TDAParam);
```

Remarks

Use the properties of TOraParam to set the value of a parameter. Objects that use parameters create TOraParam objects to represent these parameters. For example, TOraParam objects are used by TOraSQL, TCustomOraDataSet.

TOraParam shares many properties with TField, as both describe the value of a field in a dataset. However, a TField object has several properties to describe the field binding, and how the field is displayed, edited, or calculated that are not needed in a TOraParam object. Conversely, TOraParam includes properties that indicate how the field value is passed as a parameter.

Inheritance Hierarchy

TDAParam

TOraParam

See Also

- TOraDataSet
- TOraSQL
- TOraParams

Request Support DAC Forum Provide Feedback

5.14.1.17.1 Members

TOraParam class overview.

Properties

Name	Description
AsArray	Used to specify the value of the parameter when it represents the value of the Oracle array type.
AsBFile	Used to specify the value of the parameter when it represents the value of the Oracle BFile type.
AsBlob (inherited from **TDAParam**)	Used to set and read the value of the BLOB parameter as string.
AsBLOBLocator	Used to specify the value of a parameter when it represents the value of the

	BLOB type.
AsBlobRef (inherited from **TDAParam**)	Used to set and read the value of the BLOB parameter as a TBlob object.
AsCLOBLocator	Used to specify the value of the parameter when it represents the value of CLOB type.
AsCursor	Used to specify the value of the parameter when it represents the value of the PL/SQL REF CURSOR type.
AsFloat (inherited from **TDAParam**)	Used to assign the value for a float field to a parameter.
AsInteger (inherited from **TDAParam**)	Used to assign the value for an integer field to the parameter.
AsInterval	Used to specify the parameter value when it represents Oracle 9 interval type.
AsLargeInt (inherited from **TDAParam**)	Used to assign the value for a LargeInteger field to the parameter.
AsMemo (inherited from **TDAParam**)	Used to assign the value for a memo field to the parameter.
AsMemoRef (inherited from **TDAParam**)	Used to set and read the value of the memo parameter as a TBlob object.
AsNumber	Used to specify the value of the parameter when it represents the value of the Oracle internal number type.
AsObject	Used to specify the value of the parameter when it represents the value of the Oracle object type.
AsOraBlob	Used to specify the value of a parameter when it represents the value of the

	BLOB type.
AsOraClob	Used to specify the value of the parameter when it represents the value of CLOB type.
AsRef	Ulsed to specify the value of the parameter when it represents the value of the Oracle reference type.
AsSQLTimeStamp (inherited from **TDAParam**)	Used to specify the value of the parameter when it represents a SQL timestamp field.
AsString (inherited from **TDAParam**)	Used to assign the string value to the parameter.
AsTable	Used to specify the value of the parameter when it represents the value of the Oracle nested table type.
AsTimeStamp	Used to sspecify parameter value when it represents Oracle 9 timestamp type.
AsWideString (inherited from **TDAParam**)	Used to assign the Unicode string value to the parameter.
AsXML	Used to specify the value of the parameter when it represents the value of Oracle SYS.XMLTYPE.
DataType (inherited from **TDAParam**)	Indicates the data type of the parameter.
IsNull (inherited from **TDAParam**)	Used to indicate whether the value assigned to a parameter is NULL.
ItemAsDateTime	Used to access a PL/SQL table item as TDateTime by Index.
ItemAsFloat	Used to access a PL/SQL table item as Double by Index.
ItemAsInteger	Used to access a PL/SQL table item as Integer by Index.
ItemAsInterval	Used to access a PL/SQL

	table item as TOraInterval by Index.
ItemAsString	Used to access a PL/SQL table item as String by Index.
ItemAsTimeStamp	Used to access a PL/SQL table item as TOraTimeStamp by Index.
ItemIsNull	Used to indicate if an item value is Null.
ItemText	Allows to modify data without changing its type.
ItemValue	Used to access the item value as variant.
Length	Used to determine the maximum length of a PL/SQL table parameter.
National	Used to determine whether the parameter is in National charset.
ParamType (inherited from **TDAParam**)	Used to indicate the type of use for a parameter.
Size (inherited from **TDAParam**)	Specifies the size of a string type parameter.
Table	Used to determine if the parameter is a PL/SQL table.
Value (inherited from **TDAParam**)	Used to represent the value of the parameter as Variant.

Methods

Name	Description
AssignField (inherited from **TDAParam**)	Assigns field name and field value to a param.
AssignFieldValue (inherited from **TDAParam**)	Assigns the specified field properties and value to a parameter.
ItemClear	Assigns a NULL value to a table parameter item.
LoadFromFile (inherited from **TDAParam**)	Places the content of a specified file into a TDAParam object.
LoadFromStream (inherited from **TDAParam**)	Places the content from a

	stream into a TDAParam object.
SetBlobData	Sets the parameter value from the memory buffer.

Request Support DAC Forum Provide Feedback

5.14.1.17.2 Properties

Properties of the **TOraParam** class.

For a complete list of the **TOraParam** class members, see the TOraParam Members topic.

Public

Name	Description
AsArray	Used to specify the value of the parameter when it represents the value of the Oracle array type.
AsBFile	Used to specify the value of the parameter when it represents the value of the Oracle BFile type.
AsBlob (inherited from **TDAParam**)	Used to set and read the value of the BLOB parameter as string.
AsBLOBLocator	Used to specify the value of a parameter when it represents the value of the BLOB type.
AsBlobRef (inherited from **TDAParam**)	Used to set and read the value of the BLOB parameter as a TBlob object.
AsCLOBLocator	Used to specify the value of the parameter when it represents the value of CLOB type.
AsCursor	Used to specify the value of the parameter when it represents the value of the PL/SQL REF CURSOR

	type.
AsFloat (inherited from **TDAParam**)	Used to assign the value for a float field to a parameter.
AsInteger (inherited from **TDAParam**)	Used to assign the value for an integer field to the parameter.
AsInterval	Used to specify the parameter value when it represents Oracle 9 interval type.
AsLargeInt (inherited from **TDAParam**)	Used to assign the value for a LargeInteger field to the parameter.
AsMemo (inherited from **TDAParam**)	Used to assign the value for a memo field to the parameter.
AsMemoRef (inherited from **TDAParam**)	Used to set and read the value of the memo parameter as a TBlob object.
AsNumber	Used to specify the value of the parameter when it represents the value of the Oracle internal number type.
AsObject	Used to specify the value of the parameter when it represents the value of the Oracle object type.
AsOraBlob	Used to specify the value of a parameter when it represents the value of the BLOB type.
AsOraClob	Used to specify the value of the parameter when it represents the value of CLOB type.
AsRef	Ulsed to specify the value of the parameter when it represents the value of the Oracle reference type.
AsSQLTimeStamp (inherited from **TDAParam**)	Used to specify the value of the parameter when it represents a SQL timestamp field.

AsString (inherited from **TDAParam**)	Used to assign the string value to the parameter.
AsTable	Used to specify the value of the parameter when it represents the value of the Oracle nested table type.
AsTimeStamp	Used to sspecify parameter value when it represents Oracle 9 timestamp type.
AsWideString (inherited from **TDAParam**)	Used to assign the Unicode string value to the parameter.
AsXML	Used to specify the value of the parameter when it represents the value of Oracle SYS.XMLTYPE.
IsNull (inherited from **TDAParam**)	Used to indicate whether the value assigned to a parameter is NULL.
ItemAsDateTime	Used to access a PL/SQL table item as TDateTime by Index.
ItemAsFloat	Used to access a PL/SQL table item as Double by Index.
ItemAsInteger	Used to access a PL/SQL table item as Integer by Index.
ItemAsInterval	Used to access a PL/SQL table item as TOraInterval by Index.
ItemAsString	Used to access a PL/SQL table item as String by Index.
ItemAsTimeStamp	Used to access a PL/SQL table item as TOraTimeStamp by Index.
ItemIsNull	Used to indicate if an item value is Null.
ItemText	Allows to modify data without changing its type.
ItemValue	Used to access the item value as variant.

Published

Name	Description
DataType (inherited from **TDAParam**)	Indicates the data type of the parameter.
Length	Used to determine the maximum length of a PL/SQL table parameter.
National	Used to determine whether the parameter is in National charset.
ParamType (inherited from **TDAParam**)	Used to indicate the type of use for a parameter.
Size (inherited from **TDAParam**)	Specifies the size of a string type parameter.
Table	Used to determine if the parameter is a PL/SQL table.
Value (inherited from **TDAParam**)	Used to represent the value of the parameter as Variant.

See Also

- TOraParam Class
- TOraParam Class Members

Request Support DAC Forum Provide Feedback

5.14.1.17.2.1 AsArray Property

Used to specify the value of the parameter when it represents the value of the Oracle array type.

Class

TOraParam

Syntax

```
property AsArray: TOraArray;
```

Remarks

Use the AsArray property to specify the value of the parameter when it represents the value of

the Oracle array type.

Setting AsArray will set the DataType property to ftArray.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.2 AsBFile Property

Used to specify the value of the parameter when it represents the value of the Oracle BFile type.

Class

TOraParam

Syntax

```
property AsBFile: TOraFile;
```

Remarks

Use the AsBFile propery to specify the value of the parameter when it represents the value of the Oracle BFile type.

Setting AsBFile will set the DataType property to ftBFile.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.3 AsBLOBLocator Property

Used to specify the value of a parameter when it represents the value of the BLOB type.

Class

TOraParam

Syntax

```
property AsBLOBLocator: TOraLob;
```

Remarks

Use the AsBLOBLocator property to specify the value of a parameter when it represents the value of the BLOB type.

Setting AsBlobLocator will set the DataType property to ftOraBlob.

Note: This property is obsolete, use AsOraBlob instead.

See Also

- AsOraBlob

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.4 AsCLOBLocator Property

Used to specify the value of the parameter when it represents the value of CLOB type.

Class

TOraParam

Syntax

```
property AsCLOBLocator: TOraLob;
```

Remarks

Use the AsCLOBLocator property to specify the value of the parameter when it represents the value of CLOB type.

Setting AsClobLocator will set the DataType property to ftOraClob.

Note: This property is obsolete, use AsOraClob instead.

See Also

- AsOraClob

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.5 AsCursor Property

Used to specify the value of the parameter when it represents the value of the PL/SQL REF CURSOR type.

Class

TOraParam

Syntax

```
property AsCursor: TOraCursor;
```

Remarks

Use the AsCursor property to specify the value of the parameter when it represents the value of the PL/SQL REF CURSOR type.

Setting AsCursor will set the DataType property to ftCursor.

Request Support DAC Forum Provide Feedback

5.14.1.17.2.6 AsInterval Property

Used to specify the parameter value when it represents Oracle 9 interval type.

Class

TOraParam

Syntax

```
property AsInterval: TOraInterval;
```

Remarks

Use the AsInterval property to specify the parameter value when it represents Oracle 9 interval type.

Setting AsInterval will set the DataType property to ftIntervalYM or ftIntervalDS depending on the TOraInterval.DescriptorType property value.

See Also

- TOraInterval.DescriptorType

5.14.1.17.2.7 AsNumber Property

Used to specify the value of the parameter when it represents the value of the Oracle internal number type.

Class

TOraParam

Syntax

```
property AsNumber: TOraNumber;
```

Remarks

Use the AsNumber property to specify the value of the parameter when it represents the value of the Oracle internal number type.

Setting AsNumber will set the DataType property to ftNumber.

5.14.1.17.2.8 AsObject Property

Used to specify the value of the parameter when it represents the value of the Oracle object type.

Class

TOraParam

Syntax

```
property AsObject: TOraObject;
```

Remarks

Use the AsObject propery to specify the value of the parameter when it represents the value

of the Oracle object type.

Setting AsObject will set the DataType property to ftObject.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.9 AsOraBlob Property

Used to specify the value of a parameter when it represents the value of the BLOB type.

Class

TOraParam

Syntax

```
property AsOraBlob: TOraLob;
```

Remarks

Use the AsOraBlob property to specify the value of the parameter when it represents the value of BLOB type.

Setting AsOraBlob will set the DataType property to ftOraBlob.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.10 AsOraClob Property

Used to specify the value of the parameter when it represents the value of CLOB type.

Class

TOraParam

Syntax

```
property AsOraClob: TOraLob;
```

Remarks

Use the AsOraClob property to specify the value of the parameter when it represents the

value of the CLOB type.

Setting AsOraClob will set the DataType property to ftOraClob.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.11 AsRef Property

Ulsed to specify the value of the parameter when it represents the value of the Oracle reference type.

Class

TOraParam

Syntax

```
property AsRef: TOraRef;
```

Remarks

Use the AsRef property to specify the value of the parameter when it represents the value of the Oracle reference type.

Setting AsRef will set the DataType property to ftReference.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.12 AsTable Property

Used to specify the value of the parameter when it represents the value of the Oracle nested table type.

Class

TOraParam

Syntax

```
property AsTable: TOraNestTable;
```

Remarks

Use the AsTable property to specify the value of the parameter when it represents the value of the Oracle nested table type.

Setting AsTable will set the DataType property to ftTable.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.13 AsTimeStamp Property

Used to sspecify parameter value when it represents Oracle 9 timestamp type.

Class

TOraParam

Syntax

```
property AsTimeStamp: TOraTimeStamp;
```

Remarks

Specifies parameter value when it represents Oracle 9 timestamp type.

Setting AsTimeStamp will set DataType property to ftTimestamp, ftTimestampTZ or ftTimestampLTZ depending on the TOraTimeStamp.DescriptorType property value.

See Also

- TOraTimeStamp.DescriptorType

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.14 AsXML Property

Used to specify the value of the parameter when it represents the value of Oracle SYS.XMLTYPE.

Class

TOraParam

Syntax

```
property AsXML: TOraXML;
```

Remarks

Use the AsXML property to specify the value of the parameter when it represents the value of Oracle SYS.XMLTYPE.

Setting AsXML will set the DataType property to ftXML.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.15 ItemAsDateTime Property (Indexer)

Used to access a PL/SQL table item as TDateTime by Index.

Class

TOraParam

Syntax

```
property ItemAsDateTime[Index: integer]: TDateTime;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsDateTime property to access a PL/SQL table item by Index. Returns the table item as a TDateTime value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.16 ItemAsFloat Property(Indexer)

Used to access a PL/SQL table item as Double by Index.

Class

TOraParam

Syntax

```
property ItemAsFloat[Index: integer]: double;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsFloat property to access a PL/SQL table item by Index. Returns the table item as a Double value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

Request Support DAC Forum Provide Feedback

5.14.1.17.2.17 ItemAsInteger Property(Indexer)

Used to access a PL/SQL table item as Integer by Index.

Class

TOraParam

Syntax

```
property ItemAsInteger[Index: integer]: integer;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsInteger property to access a PL/SQL table item by Index. Returns the table item as a Integer value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

Request Support DAC Forum Provide Feedback

5.14.1.17.2.18 ItemAsInterval Property(Indexer)

Used to access a PL/SQL table item as TOraInterval by Index.

Class

TOraParam

Syntax

```
property ItemAsInterval[Index: integer]: TOraInterval;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsInterval property to access a PL/SQL table item by Index. Returns the table item as a TOraInterval value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

Request Support DAC Forum Provide Feedback

5.14.1.17.2.19 ItemAsString Property(Indexer)

Used to access a PL/SQL table item as String by Index.

Class

TOraParam

Syntax

```
property ItemAsString[Index: integer]: string;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsString property to access a PL/SQL table item by Index. Returns the table item as a String value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

5.14.1.17.2.20 ItemAsTimeStamp Property(Indexer)

Used to access a PL/SQL table item as TOraTimeStamp by Index.

Class

TOraParam

Syntax

```
property ItemAsTimeStamp[Index: integer]: TOraTimeStamp;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemAsTimeStamp property to access a PL/SQL table item by Index. Returns the table item as a TOraTimeStamp value.

Index starts with 1. The index value cannot be greater than the Length value.

See Also

- Table

Request Support DAC Forum Provide Feedback

5.14.1.17.2.21 ItemIsNull Property(Indexer)

Used to indicate if an item value is Null.

Class

TOraParam

Syntax

```
property ItemIsNull[Index: integer]: boolean;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemIsNull property to access PL/SQL table item by Index. Returns True, if the item value is Null.

Index starts with 1. The index value cannot be greater than Length value.

See Also

- Table

Request Support DAC Forum Provide Feedback

5.14.1.17.2.22 ItemText Property(Indexer)

Allows to modify data without changing its type.

Class

TOraParam

Syntax

```
property ItemText[Index: integer]: string;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemText property to access PL/SQL table item by Index.

Index starts with 1 and lasts till Length.

Modifying Items using ItemText doesn't affect DataType property.

Request Support DAC Forum Provide Feedback

5.14.1.17.2.23 ItemValue Property(Indexer)

Used to access the item value as variant.

Class

TOraParam

Syntax

```
property ItemValue[Index: integer]: variant;
```

Parameters

Index

Holds the index of a PL/SQL table item.

Remarks

Use the ItemValue property to access PL/SQL table item by Index. Returns the item value.

Index starts with 1. The index value cannot be greater than Length value.

See Also

- Table

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.24 Length Property

Used to determine the maximum length of a PL/SQL table parameter.

Class

TOraParam

Syntax

```
property Length: integer default 1;
```

Remarks

Use the Length property to determine the maximum length of a PL/SQL table parameter. For a scalar parameter Length is always 1.

Remember that you can use ItemAsInteger, ItemAsString and other similar properties for PL/SQL tables only.

See Also

- Table

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.25 National Property

Used to determine whether the parameter is in National charset.

Class

TOraParam

Syntax

```
property National: Boolean default False;
```

Remarks

Use the National property to determine or set whether a parameter is in National character set. When National is True, the parameter is in National character set, otherwise the parameter is in default characterset.

The parameter datatype must be one of the following: ftString, ftFixedChar, ftWideString, ftMemo or ftFixedWideChar.

Supported with Oracle 8 and higher, not supported in Direct mode.

 Request Support DAC Forum Provide Feedback

5.14.1.17.2.26 Table Property

Used to determine if the parameter is a PL/SQL table.

Class

TOraParam

Syntax

```
property Table: boolean default False;
```

Remarks

Set the Table property to True when parameter is a PL/SQL table. The maximum length of the table can be set by the Length property. Table may be ftString, ftInteger, ftFloat or ftDate type only. To access an item of a table use the ItemAsString, ItemAsInteger, ItemAsFloat, ItemAsDateTime and ItemIsNull properties.

See Also

- Length

 Request Support DAC Forum Provide Feedback

5.14.1.17.3 Methods

Methods of the **TOraParam** class.

For a complete list of the **TOraParam** class members, see the TOraParam Members topic.

Public

Name	Description
AssignField (inherited from **TDAParam**)	Assigns field name and field value to a param.
AssignFieldValue (inherited from **TDAParam**)	Assigns the specified field properties and value to a parameter.
ItemClear	Assigns a NULL value to a table parameter item.
LoadFromFile (inherited from **TDAParam**)	Places the content of a specified file into a TDAParam object.
LoadFromStream (inherited from **TDAParam**)	Places the content from a stream into a TDAParam object.
SetBlobData	Sets the parameter value from the memory buffer.

See Also

- TOraParam Class
- TOraParam Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.17.3.1 ItemClear Method

Assigns a NULL value to a table parameter item.

Class

TOraParam

Syntax

```
procedure ItemClear(Index: integer);
```

Parameters

Index

Holds the item index.

Remarks

Call the ItemClear method to assign a NULL value to a table parameter item.

Index starts with 1 and lasts till Length.

 Request Support DAC Forum Provide Feedback

5.14.1.17.3.2 SetBlobData Method

Sets the parameter value from the memory buffer.

Class

TOraParam

Syntax

```
procedure SetBlobData(Buffer: IntPtr; Size: integer);
```

Parameters

Buffer

Holds the pointer to the data.

Size

Holds the buffer size.

Remarks

Use the SetBlobData method to set the parameter value from the memory buffer. After this procedure call DataType property is assigned to ftBlob.

 Request Support DAC Forum Provide Feedback

5.14.1.18 TOraParams Class

Used to control TOraParam objects.

For a list of all members of this type, see TOraParams members.

Unit

Ora

Syntax

```
TOraParams = class(TDAParams);
```

Remarks

Use TOraParams to manage a list of TOraParam objects for an object that uses field parameters. For example, TOraStoredProc objects and TOraQuery objects use TOraParams objects to create and access their parameters.

Inheritance Hierarchy

TDAParams

TOraParams

See Also

- TOraParam
- TCustomDASQL.Params
- TCustomDADataSet.Params
- TOraDataSet.Params
- TOraSQL.Params

5.14.1.18.1 Members

TOraParams class overview.

Properties

Name	Description
Items	Used to iterate through all field parameters.

Methods

Name	Description
FindParam (inherited from **TDAParams**)	Searches for a parameter with the specified name.
ParamByName (inherited from **TDAParams**)	Searches for a parameter with the specified name.

 Request Support DAC Forum Provide Feedback

5.14.1.18.2 Properties

Properties of the **TOraParams** class.

For a complete list of the **TOraParams** class members, see the TOraParams Members topic.

Public

Name	Description
Items	Used to iterate through all field parameters.

See Also

- TOraParams Class
- TOraParams Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.18.2.1 Items Property(Indexer)

Used to iterate through all field parameters.

Class

`TOraParams`

Syntax

```
property Items[Index: integer]: TOraParam; default;
```

Parameters

Index

Holds the index in the range 0..Count - 1.

Remarks

Use the Items property to iterate through all field parameters. Index identifies the index in the range 0..Count - 1. Items can refer to a particular parameter by its index, but the TDAParams.ParamByName method is preferred to avoid depending on the order of the parameters.

Request Support DAC Forum Provide Feedback

5.14.1.19 TOraPoolingOptions Class

This class allows setting up the behaviour of the connecton pool.

For a list of all members of this type, see TOraPoolingOptions members.

Unit

Ora

Syntax

```
TOraPoolingOptions = class(TPoolingOptions);
```

Inheritance Hierarchy

TPoolingOptions

TOraPoolingOptions

Request Support DAC Forum Provide Feedback

5.14.1.19.1 Members

TOraPoolingOptions class overview.

Properties

Name	Description
ConnectionLifetime (inherited from **TPoolingOptions**)	Used to specify the maximum time during which an open connection can be used by connection pool.
MaxPoolSize (inherited from **TPoolingOptions**)	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize (inherited from **TPoolingOptions**)	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId (inherited from **TPoolingOptions**)	Used to specify an ID for a connection pool.
PoolType	Used to specify the pool type.
ProxyPassword	Used to specify a password for proxy pooling.
ProxyUsername	Ussed to specify a user name for proxy pooling.
Validate (inherited from **TPoolingOptions**)	Used for a connection to be validated when it is returned from the pool.

Request Support DAC Forum Provide Feedback

5.14.1.19.2 Properties

Properties of the **TOraPoolingOptions** class.

For a complete list of the **TOraPoolingOptions** class members, see the TOraPoolingOptions Members topic.

Published

Name	Description
ConnectionLifetime (inherited from **TPoolingOptions**)	Used to specify the maximum time during which an open connection can be used by connection pool.

MaxPoolSize (inherited from **TPoolingOptions**)	Used to specify the maximum number of connections that can be opened in connection pool.
MinPoolSize (inherited from **TPoolingOptions**)	Used to specify the minimum number of connections that can be opened in the connection pool.
PoolId (inherited from **TPoolingOptions**)	Used to specify an ID for a connection pool.
PoolType	Used to specify the pool type.
ProxyPassword	Used to specify a password for proxy pooling.
ProxyUsername	Ussed to specify a user name for proxy pooling.
Validate (inherited from **TPoolingOptions**)	Used for a connection to be validated when it is returned from the pool.

See Also

- TOraPoolingOptions Class
- TOraPoolingOptions Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.19.2.1 PoolType Property

Used to specify the pool type.

Class

TOraPoolingOptions

Syntax

```
property PoolType: TOraPoolingType default optLocal;
```

Remarks

Use the PoolType property to specify the pool type. Note that you should explicitly include

T:Devart.Odac.Units.OraConnectionPool unit to "uses" list to use the PoolType option at run-time.

 Request Support DAC Forum Provide Feedback

5.14.1.19.2.2 ProxyPassword Property

Used to specify a password for proxy pooling.

Class

TOraPoolingOptions

Syntax

```
property ProxyPassword: string;
```

Remarks

Use the Proxypassword property to specify a password for proxy pooling.

 Request Support DAC Forum Provide Feedback

5.14.1.19.2.3 ProxyUsername Property

Ussed to specify a user name for proxy pooling.

Class

TOraPoolingOptions

Syntax

```
property ProxyUsername: string;
```

Remarks

Use the ProxyUsername to specify a user name for the proxy pooling. Pool connections are stored with the same Username/Password properties. When giving connection from the pool, a connection under another user is created, based on one of these connections. Thus, connections under various users can be get from the pool.

5.14.1.20 TOraQuery Class

A component for executing queries and operating record sets. It also provides flexible way to update data.

For a list of all members of this type, see TOraQuery members.

Unit

Ora

Syntax

```
TOraQuery = class(TCustomOraQuery);
```

Remarks

TOraQuery is a direct descendant of the TOraDataSet component. It publishes most of its inherited properties and events so that they can be manipulated at design-time.

Use TOraQuery to perform fetching, insertion, deletion and update of record by dynamically generated SQL statements. TOraQuery provides automatic blocking of records, their checking before edit and refreshing after post. Set SQL, SQLInsert, SQLDelete, SQLRefresh, and SQLUpdate properties to define SQL statements for subsequent accesses to the database server. There is no restriction to their syntax, so any SQL statement is allowed. Usually you need to use INSERT, DELETE, and UPDATE statements but you also may use stored procedures in more diverse cases.

To modify records of TOraQuery SELECT statement in SQL, property should retrieve ROWID of updating table. To modify records, you can specify KeyFields. If they are not specified, TOraQuery will retrieve primary keys for UpdatingTable from metadata. TOraQuery can automatically update only one table. Updating table is defined by the UpdatingTable property if this property is set. Otherwise, the table a field of which is the first field in the field list in the SELECT clause is used as an updating table.

The SQLInsert, SQLDelete, SQLUpdate, SQLRefresh properties support automatic binding of parameters which have identical names to fields captions. To retrieve the value of a field as it was before the operation use the field name with the 'OLD_' prefix. This is especially useful

when doing field comparisons in the WHERE clause of the statement. Use the TCustomDADataSet.BeforeUpdateExecute event to assign the value to additional parameters and the TCustomDADataSet.AfterUpdateExecute event to read them.

TOraQuery performs read-only access if none of SQLInsert, SQLDelete, SQLUpdate properties is defined.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet
      TCustomOraQuery
        TOraQuery

See Also

- Updating Data with ODAC Dataset Components
- Master/Detail Relationships
- TOraStoredProc
- TOraTable

5.14.1.20.1 Members

TOraQuery class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.

ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.

FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.

LockMode	Used to specify what kind of lock will be performed when editing a record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from

	updating, inserting, or deleting data in the dataset.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when

	opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.
UpdatingTable	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the

	server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.

GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate

	method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values

	specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.

OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.14.1.20.2 Properties

Properties of the **TOraQuery** class.

For a complete list of the **TOraQuery** class members, see the TOraQuery Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.

DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a

	range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.

ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query

	component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.

UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
LockMode	Used to specify what kind of lock will be performed when editing a record.
UpdatingTable	Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

See Also

- TOraQuery Class
- TOraQuery Class Members

5.14.1.20.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TOraQuery

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

Request Support DAC Forum Provide Feedback

5.14.1.20.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TOraQuery

Syntax

```
property LockMode: TLockMode default lmNone;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

To set pessimistic locking use LockMode = lmLockImmediate, TOraDataSet.CheckMode = cmException. To set optimistic locking use LockMode = lmLockDelayed, CheckMode = cmException.

See Also

- TOraStoredProc.LockMode
- TOraTable.LockMode

 Request Support DAC Forum Provide Feedback

5.14.1.20.2.3 UpdatingTable Property

Used to specify which table in a query is assumed to be the target for subsequent data-modification queries as a result of user incentive to insert, update or delete records.

Class

TOraQuery

Syntax

```
property UpdatingTable: string;
```

Remarks

Use the UpdatingTable property to specify which table in a query is assumed to be the target for the subsequent data-modification queries as a result of user incentive to insert, update or delete records.

This property is used on Insert, Update, Delete or RefreshRecord (see also TOraDataSet.Options) if appropriate SQL (SQLInsert, SQLUpdate or SQLDelete) is not provided.

If UpdatingTable is not set then the first table used in a query is assumed to be the target.

 Request Support DAC Forum Provide Feedback

5.14.1.21 TOraReferenceField Class

A class representing an Oracle REF field in a dataset.

For a list of all members of this type, see TOraReferenceField members.

Unit

Ora

Syntax

```
TOraReferenceField = class(TReferenceField);
```

Remarks

TOraReferenceField represents an Oracle REF field in a dataset.

 Request Support DAC Forum Provide Feedback

5.14.1.21.1 Members

TOraReferenceField class overview.

Properties

Name	Description
Modified	Used to indicate whether a field was modified.

 Request Support DAC Forum Provide Feedback

5.14.1.21.2 Properties

Properties of the **TOraReferenceField** class.

For a complete list of the **TOraReferenceField** class members, see the TOraReferenceField Members topic.

Public

Name	Description
Modified	Used to indicate whether a field was modified.

See Also

- TOraReferenceField Class
- TOraReferenceField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.21.2.1 Modified Property

Used to indicate whether a field was modified.

Class

TOraReferenceField

Syntax

```
property Modified: boolean;
```

Remarks

Use the Modify property to indicate whether a field was modified. The property is writable.

Request Support DAC Forum Provide Feedback

5.14.1.22 TOraSession Class

A component for maintaining connection to an Oracle database.

For a list of all members of this type, see TOraSession members.

Unit

Ora

Syntax

```
TOraSession = class(TCustomDAConnection);
```

Remarks

The TOraSession component is used to maintain connection to an Oracle database. After setting the Username, Password and Server properties, you can establish a connection to the database by calling the Connect method or setting the Connected property to True. There are also many properties at the session level that affect the default behavior of the queries executed within this session. Furthermore, you can control transactions using methods from this class.

All components that are dedicated to perform data access, such as TOraQuery, TOraSQL, TOraScript, must have their Session property assigned with one of the TOraSession instances.

Inheritance Hierarchy

TCustomDAConnection

TOraSession

See Also

- TOraDataSet.Session
- TOraSQL.Session

Request Support DAC Forum Provide Feedback

5.14.1.22.1 Members

TOraSession class overview.

Properties

Name	Description
AutoCommit	Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
Connected	Used to indicate if the database connection is active.
ConnectMode	Used to specify the system privileges to use when a user connects to the server.
ConnectPrompt	Used to supply a prompt for a name and password.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the

	connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
Debug	Used to display SQL statements being executed with their parameter values and data types.
Home	Used to specify the Oracle client to be used in the application.
HomeName	Used to select the Oracle client to use with the application.
HttpOptions	Used to set up the HTTP options.
InternalName	Used to get or set the client database name that will be recorded when performing global transactions.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
LastError	Used to get an error code which resulted from previous call to the OCI interface function.
LDA	Provides a pointer to Oracle 7 login data area of the current connection.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
OCICallStyle	Indicates the set of OCI routines used.
OCISvcCtx	Used to return Oracle 8 service context handle of the current connection.
Options	Used to specify the behaviour of a TOraSession object.
OracleVersion	Used to get Oracle server

	version number as string.
Password	Used to specify a password for a connection.
Pooling (inherited from **TCustomDAConnection**)	Enables or disables using connection pool.
PoolingOptions	Used to specify the behaviour of connection pool.
ProxySession	Used to enable multiple user sessions within a single database session.
Schema	Used to change the current schema of the session to the specified schema.
Server	Contains the server name.
SQL	Uses embedded TOraSQL object to execute any SQL statement.
SSLOptions	Used to set up the SSL options.
ThreadSafety	Used to allow the usage of the OCI in multi-threaded environment.
Username	Contains username.

Methods

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TOraSession components.
AssignLDA	The method is used to assign LDA handle.
AssignSvcCtx	Overloaded. The method is used to assign service context handle.
ChangePassword	Changes the password of an account to the new password.

Commit (inherited from **TCustomDAConnection**)	Commits current transaction.
Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateSQL (inherited from **TCustomDAConnection**)	Creates a component for queries execution.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
ExecProc (inherited from **TCustomDAConnection**)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from **TCustomDAConnection**)	Allows to execute a stored procedure or function.
ExecSQL (inherited from **TCustomDAConnection**)	Executes a SQL statement with parameters.
ExecSQLEx (inherited from **TCustomDAConnection**)	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames (inherited from **TCustomDAConnection**)	Returns a database list from the server.
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetSequenceNames	Provides the names of available sequences.
GetStoredProcNames (inherited from **TCustomDAConnection**)	Returns a list of stored procedures from the server.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
ParamByName	Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.

Ping (inherited from **TCustomDAConnection**)	Used to check state of connection to the server.
RemoveFromPool (inherited from **TCustomDAConnection**)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction	Overloaded. Begins a new user transaction against the database server.

Events

Name	Description
OnConnectChange	Occurs after Connected property was changed.
OnConnectionLost (inherited from **TCustomDAConnection**)	This event occurs when connection was lost.
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.
OnFailover	Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.
OnInfoMessage	The event occurs if the server returned the OCI_SUCCESS_WITH_INFO error.

5.14.1.22.2 Properties

Properties of the **TOraSession** class.

For a complete list of the **TOraSession** class members, see the TOraSession Members topic.

Public

Name	Description
ConnectDialog (inherited from **TCustomDAConnection**)	Allows to link a TCustomConnectDialog component.
ConnectString (inherited from **TCustomDAConnection**)	Used to specify the connection information, such as: UserName, Password, Server, etc.
ConvertEOL (inherited from **TCustomDAConnection**)	Allows customizing line breaks in string fields and parameters.
InternalName	Used to get or set the client database name that will be recorded when performing global transactions.
InTransaction (inherited from **TCustomDAConnection**)	Indicates whether the transaction is active.
LastError	Used to get an error code which resulted from previous call to the OCI interface function.
LDA	Provides a pointer to Oracle 7 login data area of the current connection.
LoginPrompt (inherited from **TCustomDAConnection**)	Specifies whether a login dialog appears immediately before opening a new connection.
OCICallStyle	Indicates the set of OCI routines used.
OCISvcCtx	Used to return Oracle 8 service context handle of the current connection.
OracleVersion	Used to get Oracle server version number as string.

Pooling (inherited from **TCustomDAConnection**)	Enables or disables using connection pool.
ProxySession	Used to enable multiple user sessions within a single database session.
SQL	Uses embedded TOraSQL object to execute any SQL statement.

Published

Name	Description
AutoCommit	Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.
Connected	Used to indicate if the database connection is active.
ConnectMode	Used to specify the system privileges to use when a user connects to the server.
ConnectPrompt	Used to supply a prompt for a name and password.
Debug	Used to display SQL statements being executed with their parameter values and data types.
Home	Used to specify the Oracle client to be used in the application.
HomeName	Used to select the Oracle client to use with the application.
HttpOptions	Used to set up the HTTP options.
Options	Used to specify the behaviour of a TOraSession object.
Password	Used to specify a password for a connection.

PoolingOptions	Used to specify the behaviour of connection pool.
Schema	Used to change the current schema of the session to the specified schema.
Server	Contains the server name.
SSLOptions	Used to set up the SSL options.
ThreadSafety	Used to allow the usage of the OCI in multi-threaded environment.
Username	Contains username.

See Also

- TOraSession Class
- TOraSession Class Members

Request Support DAC Forum Provide Feedback

5.14.1.22.2.1 AutoCommit Property

Used to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server.

Class

TOraSession

Syntax

```
property AutoCommit: boolean;
```

Remarks

Use the AutoCommit property to permit or prevent permanent updates, insertions, and deletions of data associated with the current transaction against the database server without explicit calls to the Commit or Rollback methods.

Set AutoCommit to True to permit implicit call to Commit method after every database

access.

AutoCommit property in TOraSession has higher precedence over the same properties in dataset components. Its default value is True.

Note: The AutoCommit property in TOraSesion globally specifies whether all queries to modify a database are implicitly committed or not. Components which descend from the TCustomDADataSet and TCustomDASQL classes inherit their AutoCommit properties. This allows them to specify their implicit transaction selectively committing the behavior after each data modifying access.

This is an example of procedure that removes all records from Dept table and makes this change permanent.

Example

```
procedure TForm1.DeleteClick(Sender: TObject);
begin
   OraSQL.Session := OraSession;
   OraSession.AutoCommit := True;
   OraSQL.AutoCommit := False;
   OraSQL.SQL := 'DELETE FROM Dept';
   OraSQL.Execute;          // delete all records, commit is not performed
   OraSession.Rollback;// restore deleted records
   OraSession.AutoCommit := False;
   OraSQL.AutoCommit := True;
   OraSQL.SQL := 'DELETE FROM Dept';
   OraSQL.Execute;          // delete all records, commit is not performed
   OraSession.Rollback; // restore deleted records
   OraSession.AutoCommit := True;
   OraSQL.AutoCommit := True;
   OraSQL.SQL := 'DELETE FROM Dept';
   OraSQL.Execute;          // delete all records, commit is performed
   OraSession.Rollback; // couldn't restore deleted records
end;
```

See Also

- TCustomDAConnection.Commit

5.14.1.22.2.2 Connected Property

Used to indicate if the database connection is active.

Class

TOraSession

Syntax

```
property Connected stored IsConnectedStored;
```

Remarks

Use the Connected property to indicate whether the database connection is active. Setting this property is equivalent to calling the TCustomDAConnection.Connect or TCustomDAConnection.Disconnect methods at runtime.

OnConnectChange event occurs after the Connected property has been changed.

See Also

- TCustomDAConnection.Connect
- TCustomDAConnection.Disconnect
- OnConnectChange

5.14.1.22.2.3 ConnectMode Property

Used to specify the system privileges to use when a user connects to the server.

Class

TOraSession

Syntax

```
property ConnectMode: TConnectMode default DefValConnectMode;
```

Remarks

Use the ConnectMode property to specify which system privileges to use when a user

connects to the server.

Note: User must have SYSOPER, SYSDBA or both these roles granted before he connects to the server and wishes to use either of these roles. ConnectMode is not supported for OCI 7.

See Also

- Password
- Server
- Username

Request Support DAC Forum Provide Feedback

5.14.1.22.2.4 ConnectPrompt Property

Used to supply a prompt for a name and password.

Class

TOraSession

Syntax

```
property ConnectPrompt: boolean stored False default True;
```

Remarks

Set the ConnectPrompt property to True to provide login support when establishing a connection. When ConnectPrompt is True, a dialog appears to prompt a user for a name and a password.

When ConnectPrompt is False, an application must supply user name and password values programmatically.

Warning: Storing a hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

See Also

- Password

- Server
- Username
- TCustomDAConnection.ConnectString

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.5 Debug Property

Used to display SQL statements being executed with their parameter values and data types.

Class

TOraSession

Syntax

```
property Debug: boolean;
```

Remarks

Set the Debug property to True to display SQL statements being executed with their parameter values and data types.

Note: To use this property you should explicitly include OdacVcl (OdacClx under Linux) unit to your project.

If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.6 Home Property

Used to specify the Oracle client to be used in the application.

Class

TOraSession

Syntax

```
property Home: TOracleHome stored False;
```

Remarks

Set the Home property to select which Oracle client will be used in your application. Use this property in cases when there is a number of Oracle clients on the machine. ODAC searches all available homes in the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ALL_HOMES registry folder.

See Also

- TCustomDAConnection.Connect
- TOracleHome

Request Support DAC Forum Provide Feedback

5.14.1.22.2.7 HomeName Property

Used to select the Oracle client to use with the application.

Class

TOraSession

Syntax

```
property HomeName: string;
```

Remarks

Use the HomeName property to select which Oracle client will be used in your application. Use this property in cases when there is a number of Oracle clients on the machine. ODAC searches all available homes in HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE registry folder. If HomeName property is set to '', ODAC uses first directory from the list of homes encountered in environment PATH variable as default Oracle home.

See Also

- TCustomDAConnection.Connect
- TOracleHome

5.14.1.22.2.8 HttpOptions Property

Used to set up the HTTP options.

Class

TOraSession

Syntax

```
property HttpOptions: THttpOptions;
```

Remarks

Used the HttpOptions property to set up the HTTP options.

5.14.1.22.2.9 InternalName Property

Used to get or set the client database name that will be recorded when performing global transactions.

Class

TOraSession

Syntax

```
property InternalName: string;
```

Remarks

Use the InternalName property to get or set the client database name that will be recorded when performing global transactions. While there is no actual global transaction support, setting this property to a non-empty string can give performance gains on SQL statement execution. But there is one undesirable effect: you cannot commit or rollback transaction from PL/SQL block. You should call TCustomDAConnection.Commit or TCustomDAConnection.Rollback explicitly.

See Also

- TCustomDAConnection.Commit
- TCustomDAConnection.Rollback

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.10 LastError Property

Used to get an error code which resulted from previous call to the OCI interface function.

Class

TOraSession

Syntax

```
property LastError: integer;
```

Remarks

Use the LastError property to get an error code which resulted from previous call to the OCI interface function.

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.11 LDA Property

Provides a pointer to Oracle 7 login data area of the current connection.

Class

TOraSession

Syntax

```
property LDA: PLDA;
```

Remarks

Call the LDA method to get a pointer to Oracle 7 login data area of the current connection.

LDA structure is relevant mainly to OCI 7 call interface.

See Also

- OCISvcCtx
- AssignConnect

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.12 OCICallStyle Property

Indicates the set of OCI routines used.

Class

TOraSession

Syntax

```
property OCICallStyle: TOCICallStyle;
```

Remarks

Use the OCICallStyle property to check what set of OCI routines is used. Write OCICallStyle before connection to specify that either OCI 7.3 or OCI 8.0 routines will be used. TOraSession initializes this property on behalf of the default OCI client found in the system at the time when OCI library is being loaded.

See Also

- Options

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.13 OCISvcCtx Property

Used to return Oracle 8 service context handle of the current connection.

Class

TOraSession

Syntax

```
property OCISvcCtx: TOCISvcCtx;
```

Remarks

Use the OCISvcCtx property to return Oracle 8 service context handle of the current connection.

See Also

- LDA
- AssignConnect

Request Support DAC Forum Provide Feedback

5.14.1.22.2.14 Options Property

Used to specify the behaviour of a TOraSession object.

Class

TOraSession

Syntax

```
property Options: TOraSessionOptions;
```

Remarks

Set the properties of Options to specify the behaviour of a TOraSession object.

Descriptions of all options are in the table below.

Option Name	Description
CharLength	Used to specify the size of a single character in bytes.
Charset	Used to set the character set that ODAC uses to read and write character data.
ClientIdentifier	Used to determine the client identifier in the session.
ConnectionTimeout	Used to specify the time to wait for a

	connection to open before raising an exception.
ConvertEOL	Affects the line break behavior in string fields and parameters.
DateFormat	Used to specify the default date format used when Oracle makes conversions from internal date format into string values and vice versa.
DateLanguage	Used to specify the default language used when Oracle parses internal date format into string values and vice versa.
Direct	Used for ODAC to connect directly over TCP/IP (in Direct mode) and without requiring Oracle software on the client side.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
EnableIntegers	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.
EnableLargeint	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.
EnableNumbers	Used for ODAC to map Oracle numbers with precision larger than 15 to TOraNumberField.
EnableOraTimestamp	Used to create TOraTimeStampField for columns of TIMESTAMP data type.
IPVersion	Used to specify Internet Protocol Version.
OptimizerMode	Used to get or set the default optimizer mode for connection.
StatementCache	Used for ODAC to cache statement handles.
StatementCacheSize	Used to specify the statement handle cache size.
SubscriptionPort	Sets the client port used to receive notifications.
UnicodeEnvironment	Enables or disables using OCI Unicode Environment.
UseOCI7	Used to force TOraSession use OCI 7 call style only.
UseUnicode	Used to enable or disable Unicode support.

See Also

- Connecting in Direct Mode
- Unicode Character Data
- TOraNumberField
- TOraDataSet.Options
- TOraSQL.StatementCache

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.15 OracleVersion Property

Used to get Oracle server version number as string.

Class

TOraSession

Syntax

```
property OracleVersion: string;
```

Remarks

Use the OracleVersion property to get Oracle server version number as string, for a example '7.3.2.3'

Works only when TOraSession instance is connected.

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.16 Password Property

Used to specify a password for a connection.

Class

TOraSession

Syntax

```
property Password: string;
```

Remarks

Use the Password property to specify a password for a connection. TOraSession uses Password to build connect string in the form **Username/Password@Server** .

When property is being changed TOraSession calls Disconnect method

See Also

- Username
- Server
- TCustomDAConnection.Connect

Request Support DAC Forum Provide Feedback

5.14.1.22.2.17 PoolingOptions Property

Used to specify the behaviour of connection pool.

Class

TOraSession

Syntax

```
property PoolingOptions: TOraPoolingOptions;
```

Remarks

Set the properties of PoolingOptions to specify the behaviour of the connection pool.

Descriptions of all options are in the table below.

Option Name	Description
PoolType	Used to specify the pool type.
ProxyPassword	Used to specify a password for proxy pooling.
ProxyUsername	Ussed to specify a user name for proxy

	pooling.

See Also

- TCustomDAConnection.Pooling

Request Support DAC Forum Provide Feedback

5.14.1.22.2.18 ProxySession Property

Used to enable multiple user sessions within a single database session.

Class

TOraSession

Syntax

```
property ProxySession: TOraSession;
```

Remarks

Applications can have multiple user sessions within a single database session. These "lightweight sessions" allow each user to be authenticated, preserving the identity of the real user through the middle tier.

The application server creates a proxy session for itself once it connects to a server. It authenticates itself to the database in a normal way creating the application server trust zone. The application server identity is now well known and trusted to the data server. Application server verifies the identity of a client. After that it can create TOraSession component and establish session for each client without authentication on Oracle server. That will reduce time for connection.

For each client session you must refer the TOraSession.ProxySession property to proxy TOraSession object. TOraSession.Password may by empty for client session. To use this feature Oracle users must have CONNECT THROUGH privilege.

Note: ProxySession property in TOraSession is supported with OCI connection only when Options.Direct=False

Request Support DAC Forum Provide Feedback

Reserved.

5.14.1.22.2.19 Schema Property

Used to change the current schema of the session to the specified schema.

Class

TOraSession

Syntax

```
property Schema: string stored IsSchemaStored;
```

Remarks

Use the Schema property to change the current schema of the session to the specified schema. This setting offers a convenient way to perform operations on objects in a schema other than that of the current user without having to qualify the objects with the schema name. This setting changes the current schema, but it neither changes the session user or the current user, nor gives you any additional system or object privileges for the session.

If Connected = True read this property to receive the name of the current schema.

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.20 Server Property

Contains the server name.

Class

TOraSession

Syntax

```
property Server: string;
```

Remarks

Use the Server property to supply server name to handle server's request for a login.

Formatting of this property is different and depends on the value of Options.Direct property:

If Options. Direct is set to False, Server assumes **TNS** alias name for the requested database or an **EZCONNECT** connection string.

If Options.Direct is True, Server accepts a string holding three fields separated by a colon. If Oracle servers has SID is equal to Service Name then string is the following: "Host:Port:SID". If Oracle Server has SID differ from Service Name then string is the following: "Host:Port:sid=SID" for connection using SID and "Host:Port:sn=Service Name" for connection using Service Name. Here Host is an IP address of the server that hosts the database, Port is a port number that server listens, SID is a system identifier that specifies an Oracle database instance name, and Service Name is a system alias to an Oracle database instance (or many instances).

Note: EZCONNECT connection string format is supported in Direct Mode as well.

Note: If prefixes sid= or sn= aren't set, then connection will be established using SID.

See Also

- Username
- Password
- TCustomDAConnection.Connect
- HomeName
- Connecting in Direct Mode

Request Support DAC Forum Provide Feedback

5.14.1.22.2.21 SQL Property

Uses embedded TOraSQL object to execute any SQL statement.

Class

TOraSession

Syntax

```
property SQL: TOraSQL;
```

Remarks

You can use embedded TOraSQL object to execute any SQL statement.

See Also

- TOraSQL
- TCustomDAConnection.ExecSQLEx
- ParamByName

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.22 SSLOptions Property

Used to set up the SSL options.

Class

TOraSession

Syntax

```
property SSLOptions: TOraConnectionSSLOptions;
```

Remarks

Use the SSLOptions property to set up the SSL options.

 Request Support DAC Forum Provide Feedback

5.14.1.22.2.23 ThreadSafety Property

Used to allow the usage of the OCI in multi-threaded environment.

Class

TOraSession

Syntax

```
property ThreadSafety: boolean default True;
```

Remarks

Use the ThreadSafety property to enable the usage of the OCI in multi-threaded environment. The ThreadSafety property must be True before any non-blocking fetch of rows or SQL statement execution takes place.

See Also

- TOraDataSet.NonBlocking
- TOraSQL.NonBlocking

Request Support DAC Forum Provide Feedback

5.14.1.22.2.24 Username Property

Contains username.

Class

TOraSession

Syntax

```
property Username: string;
```

Remarks

Use the Username property to supply user name to handle server's request for a login.

TOraSession uses the Username, Password and Server properties to build connect string in the format **Username/Password@Server** .

When property is being changed TOraSession calls Disconnect method

See Also

- Password
- Server
- TCustomDAConnection.Connect

Request Support DAC Forum Provide Feedback

5.14.1.22.3 Methods

Methods of the **TOraSession** class.

For a complete list of the **TOraSession** class members, see the TOraSession Members topic.

Public

Name	Description
ApplyUpdates (inherited from **TCustomDAConnection**)	Overloaded. Applies changes in datasets.
AssignConnect	Shares database connection between the TOraSession components.
AssignLDA	The method is used to assign LDA handle.
AssignSvcCtx	Overloaded. The method is used to assign service context handle.
ChangePassword	Changes the password of an account to the new password.
Commit (inherited from **TCustomDAConnection**)	Commits current transaction.
Connect (inherited from **TCustomDAConnection**)	Establishes a connection to the server.
CreateSQL (inherited from **TCustomDAConnection**)	Creates a component for queries execution.
Disconnect (inherited from **TCustomDAConnection**)	Performs disconnect.
ExecProc (inherited from **TCustomDAConnection**)	Allows to execute stored procedure or function providing its name and parameters.
ExecProcEx (inherited from **TCustomDAConnection**)	Allows to execute a stored procedure or function.
ExecSQL (inherited from **TCustomDAConnection**)	Executes a SQL statement with parameters.
ExecSQLEx (inherited from **TCustomDAConnection**)	Executes any SQL statement outside the TQuery or TSQL components.
GetDatabaseNames (inherited from	Returns a database list from the server.

TCustomDAConnection)	
GetKeyFieldNames (inherited from **TCustomDAConnection**)	Provides a list of available key field names.
GetSequenceNames	Provides the names of available sequences.
GetStoredProcNames (inherited from **TCustomDAConnection**)	Returns a list of stored procedures from the server.
GetTableNames (inherited from **TCustomDAConnection**)	Provides a list of available tables names.
MonitorMessage (inherited from **TCustomDAConnection**)	Sends a specified message through the TCustomDASQLMonitor component.
ParamByName	Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.
Ping (inherited from **TCustomDAConnection**)	Used to check state of connection to the server.
RemoveFromPool (inherited from **TCustomDAConnection**)	Marks the connection that should not be returned to the pool after disconnect.
Rollback (inherited from **TCustomDAConnection**)	Discards all current data changes and ends transaction.
RollbackToSavepoint	Cancels all updates for the current transaction.
Savepoint	Defines a point in the transaction to which you can roll back later.
StartTransaction	Overloaded. Begins a new user transaction against the database server.

See Also

- TOraSession Class

- TOraSession Class Members

5.14.1.22.3.1 AssignConnect Method

Shares database connection between the TOraSession components.

Class

TOraSession

Syntax

```
procedure AssignConnect(Source: TOraSession); overload;
```

Parameters

Source

Points to a preconnected session and sets Connected property to True for this instance of TOraSession.

Remarks

Use the AssignConnect method to share database connection between the TOraSession components.

AssignConnect assumes that the Source parameter points to a preconnected session and sets Connected property to True for this instance of TOraSession. Note that AssignConnect doesn't make any references to the Source session. So before disconnecting parent session call AssignConnect(Nil) or Disconnect method for all assigned sessions.

Example

```
OraSession1.Connect;
OraSession2.AssignConnect(OraSession1);
                                    // OraSession2.Connected is True
OraSQL.Session := OraSession2;
OraSQL.Execute;
OraSession2.AssignConnect(nil);
                                    // OraSession2.Connected is False
OraSession1.Disconnect;
```

See Also

- LDA

- OCISvcCtx
- TCustomDAConnection.Connect

5.14.1.22.3.2 AssignLDA Method

The method is used to assign LDA handle.

Class

TOraSession

Syntax

```
procedure AssignLDA(LDA: pLDA);
```

Parameters

LDA

Specifies the LDA handle.

5.14.1.22.3.3 AssignSvcCtx Method

The method is used to assign service context handle.

Class

TOraSession

Overload List

Name	Description
AssignSvcCtx(hOCISvcCtx: pOCISvcCtx)	The method is used to assign service context handle.
AssignSvcCtx(hOCISvcCtx: pOCISvcCtx; hOCIEnv: pOCIEnv)	The method is used to assign service context handle.

The method is used to assign service context handle.

Class

TOraSession

Syntax

```
procedure AssignSvcCtx(hOCISvcCtx: pOCISvcCtx); overload;
deprecated;
```

Parameters

hOCISvcCtx

Specifies the service context handle.

 Request Support DAC Forum Provide Feedback

The method is used to assign service context handle.

Class

TOraSession

Syntax

```
procedure AssignSvcCtx(hOCISvcCtx: pOCISvcCtx; hOCIEnv: pOCIEnv);
overload;
```

Parameters

hOCISvcCtx

Specifies the service context handle.

hOCIEnv

Specifies the environment handle.

 Request Support DAC Forum Provide Feedback

5.14.1.22.3.4 ChangePassw ord Method

Changes the password of an account to the new password.

Class

TOraSession

Syntax

```
procedure ChangePassword(NewPassword: string);
```

Parameters

NewPassword

Takes the new password.

Remarks

Call the ChangePassword method to replace the current password of an account with the new password.

The previous values must be provided for the Password and UserName properties before calling ChangePassword.

The ChangePassword method is used mainly when logging in to the user account fails due to an expired password or any other reason accompanied by an exception with ORA-2800 Oracle error code family (see Oracle Error Messages).

See Also

- Username
- Password

5.14.1.22.3.5 GetSequenceNames Method

Provides the names of available sequences.

Class

TOraSession

Syntax

```
procedure GetSequenceNames(List: TStrings; AllSequences: boolean =
False);
```

Parameters

List
Holds the list of available sequences names.

AllSequences
If True, method returns sequences from all schemas.

Remarks

Call the GetSequenceNames method to get the names of available sequences.

See Also

- TCustomDAConnection.GetTableNames
- TCustomDAConnection.GetStoredProcNames

 Request Support DAC Forum Provide Feedback

5.14.1.22.3.6 ParamByName Method

Provides access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc.

Class

TOraSession

Syntax

```
function ParamByName(Name: string): TOraParam;
```

Parameters

Name
Holds the parameter name.

Return Value
a TOraParam object.

Remarks

Use the ParamByName method to get access to OUT parameters and their values after processing SQL statement with ExecSQL or stored procedure with ExecProc. Name should be equal to the parameter name as it occurred in SQL statement.

Implicitely calls ParamByName function of TOraSQL.

See Also

- SQL
- TCustomDAConnection.ExecSQL
- TCustomDAConnection.ExecSQLEx

 Request Support DAC Forum Provide Feedback

5.14.1.22.3.7 RollbackToSavepoint Method

Cancels all updates for the current transaction.

Class

TOraSession

Syntax

```
procedure RollbackToSavepoint(const Name: string);
```

Parameters

Name

Remarks

Call the RollbackToSavepoint method to cancel all updates for the current transaction and restore its state up to the moment of the last defined savepoint.

See Also

- Savepoint
- TCustomDAConnection.Rollback

 Request Support DAC Forum Provide Feedback

5.14.1.22.3.8 Savepoint Method

Defines a point in the transaction to which you can roll back later.

Class

TOraSession

Syntax

```
procedure Savepoint(const Name: string);
```

Parameters

Name

Remarks

Call the Savepoint method to define a point in the transaction to which you can roll back later. As the parameter, you can pass any valid name to identify the savepoint.

To roll back to the last savepoint call RollbackToSavepoint.

See Also

- RollbackToSavepoint

5.14.1.22.3.9 StartTransaction Method

Begins a new user transaction against the database server.

Class

TOraSession

Overload List

Name	Description
StartTransaction	Begins a new user transaction against the database server.
StartTransaction(IsolationLevel: TOraIsolationLevel; **const** RollbackSegment: **string**; **const** Name: **string**)	Begins a new user transaction against the database server.

Begins a new user transaction against the database server.

Class

TOraSession

Syntax

```
procedure StartTransaction; overload; override;
```

Remarks

StartTransaction is an overload method for TCustomDAConnection.StartTransaction. Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling TCustomDAConnection.Commit or TCustomDAConnection.Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback to cancel them.

See Also

- TCustomDAConnection.Commit
- TCustomDAConnection.Rollback
- TCustomDAConnection.InTransaction
- TCustomDAConnection.StartTransaction

Begins a new user transaction against the database server.

Class

TOraSession

Syntax

```
procedure StartTransaction(IsolationLevel: TOraIsolationLevel;
const RollbackSegment: string = ''; const Name: string = '');
reintroduce; overload;
```

Parameters

IsolationLevel

Specifies how the transactions containing database modifications are handled.

RollbackSegment

Holds the rollback segment to assign the current transaction to.

Name

Holds the current transaction name.

Remarks

Specify the RollbackSegment parameter to assign the current transaction to the specified rollback segment. This clause also implicitly establishes the transaction as a read/write transaction.

The Name parameter is useful in distributed database environments when you must identify and resolve in-doubt transactions. The text string is limited to 255 bytes.

Request Support DAC Forum Provide Feedback

5.14.1.22.4 Events

Events of the **TOraSession** class.

For a complete list of the **TOraSession** class members, see the TOraSession Members topic.

Public

Name	Description
OnConnectionLost (inherited from **TCustomDAConnection**)	This event occurs when connection was lost.
OnError (inherited from **TCustomDAConnection**)	This event occurs when an error has arisen in the connection.

Published

Name	Description
OnConnectChange	Occurs after Connected property was changed.
OnFailover	Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.
OnInfoMessage	The event occurs if the server returned the OCI_SUCCESS_WITH_INFO error.

See Also

- TOraSession Class
- TOraSession Class Members

Request Support DAC Forum Provide Feedback

5.14.1.22.4.1 OnConnectChange Event

Occurs after Connected property was changed.

Class

TOraSession

Syntax

```
property OnConnectChange: TConnectChangeEvent;
```

Remarks

Occurs after the Connected property was changed. Connected parameter indicates whether the connection is active or not.

Note: This event is obsolete. Use AfterConnect and AfterDisconnect event handlers instead.

See Also

- TCustomDAConnection.Connect

- TCustomDAConnection.Disconnect

5.14.1.22.4.2 OnFailover Event

Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.

Class

TOraSession

Syntax

```
property OnFailover: TFailoverEvent;
```

Remarks

Occurs when Transparent Application Failover (TAF) seamlessly attempts to failover to another Oracle instance.

FailoverType parameter specifies the type of failover. This allows the event to know what type of failover the client has requested.

See Also

- Transparent Application Failover Support

5.14.1.22.4.3 OnInfoMessage Event

The event occurs if the server returned the OCI_SUCCESS_WITH_INFO error.

Class

TOraSession

Syntax

```
property OnInfoMessage: TInfoMessageEvent;
```


5.14.1.23 TOraSessionOptions Class

This class allows setting up the behaviour of the TOraSession class.

For a list of all members of this type, see TOraSessionOptions members.

Unit

Ora

Syntax

```
TOraSessionOptions = class(TDAConnectionOptions);
```

Inheritance Hierarchy

TDAConnectionOptions

TOraSessionOptions

5.14.1.23.1 Members

TOraSessionOptions class overview.

Properties

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
CharLength	Used to specify the size of a single character in bytes.
Charset	Used to set the character set that ODAC uses to read and write character data.
ClientIdentifier	Used to determine the client identifier in the session.
ConnectionTimeout	Used to specify the time to

	wait for a connection to open before raising an exception.
ConvertEOL	Affects the line break behavior in string fields and parameters.
DateFormat	Used to specify the default date format used when Oracle makes conversions from internal date format into string values and vice versa.
DateLanguage	Used to specify the default language used when Oracle parses internal date format into string values and vice versa.
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
Direct	Used for ODAC to connect directly over TCP/IP (in Direct mode) and without requiring Oracle software on the client side.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
EnableIntegers	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.

EnableLargeint	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.
EnableNumbers	Used for ODAC to map Oracle numbers with precision larger than 15 to TOraNumberField.
EnableOraTimestamp	Used to create TOraTimeStampField for columns of TIMESTAMP data type.
IPVersion	Used to specify Internet Protocol Version.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.
OptimizerMode	Used to get or set the default optimizer mode for connection.
StatementCache	Used for ODAC to cache statement handles.
StatementCacheSize	Used to specify the statement handle cache size.
SubscriptionPort	Sets the client port used to receive notifications.
UnicodeEnvironment	Enables or disables using OCI Unicode Environment.
UseOCI7	Used to force TOraSession use OCI 7 call style only.
UseUnicode	Used to enable or disable Unicode support.

5.14.1.23.2 Properties

Properties of the **TOraSessionOptions** class.

For a complete list of the **TOraSessionOptions** class members, see the TOraSessionOptions Members topic.

Public

Name	Description
DefaultSortType (inherited from **TDAConnectionOptions**)	Used to determine the default type of local sorting for string fields. It is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.
DisconnectedMode (inherited from **TDAConnectionOptions**)	Used to open a connection only when needed for performing a server call and closes after performing the operation.
KeepDesignConnected (inherited from **TDAConnectionOptions**)	Used to prevent an application from establishing a connection at the time of startup.
LocalFailover (inherited from **TDAConnectionOptions**)	If True, the TCustomDAConnection.OnConnectionLost event occurs and a failover operation can be performed after connection breaks.

Published

Name	Description
AllowImplicitConnect (inherited from **TDAConnectionOptions**)	Specifies whether to allow or not implicit connection opening.
CharLength	Used to specify the size of a single character in bytes.
Charset	Used to set the character set that ODAC uses to read and

	write character data.
ClientIdentifier	Used to determine the client identifier in the session.
ConnectionTimeout	Used to specify the time to wait for a connection to open before raising an exception.
ConvertEOL	Affects the line break behavior in string fields and parameters.
DateFormat	Used to specify the default date format used when Oracle makes conversions from internal date format into string values and vice versa.
DateLanguage	Used to specify the default language used when Oracle parses internal date format into string values and vice versa.
Direct	Used for ODAC to connect directly over TCP/IP (in Direct mode) and without requiring Oracle software on the client side.
EnableBCD	Used to enable currency type. Default value of this option is False.
EnableFMTBCD	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
EnableIntegers	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.
EnableLargeint	Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.
EnableNumbers	Used for ODAC to map Oracle numbers with precision larger than 15 to TOraNumberField.
EnableOraTimestamp	Used to create

	TOraTimeStampField for columns of TIMESTAMP data type.
IPVersion	Used to specify Internet Protocol Version.
OptimizerMode	Used to get or set the default optimizer mode for connection.
StatementCache	Used for ODAC to cache statement handles.
StatementCacheSize	Used to specify the statement handle cache size.
SubscriptionPort	Sets the client port used to receive notifications.
UnicodeEnvironment	Enables or disables using OCI Unicode Environment.
UseOCI7	Used to force TOraSession use OCI 7 call style only.
UseUnicode	Used to enable or disable Unicode support.

See Also

- TOraSessionOptions Class
- TOraSessionOptions Class Members

Request Support DAC Forum Provide Feedback

5.14.1.23.2.1 CharLength Property

Used to specify the size of a single character in bytes.

Class

TOraSessionOptions

Syntax

```
property CharLength: TCharLength default 0;
```

Remarks

Use the CharLength property to specify the size of a single character in bytes. Set this option with the number in range [0..6] to reflect Oracle support for the national languages. Setting CharLength to zero will instruct TOraSession to interrogate Oracle server for the actual character length. The default value is 1.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.2 Charset Property

Used to set the character set that ODAC uses to read and write character data.

Class

TOraSessionOptions

Syntax

```
property Charset: string;
```

Remarks

Use the Charset property to set the character set that ODAC uses to read and write character data. Supported with Oracle 8 client or later.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.3 ClientIdentifier Property

Used to determine the client identifier in the session.

Class

TOraSessionOptions

Syntax

```
property ClientIdentifier: string;
```

Remarks

Use the ClientIdentifier property to determine the client identifier in the session. The client

identifier can be set in the session handle at any time during the session. Then, on the next request to the server, the information is propagated and stored in the server session. The first character of the ClientIdentifier should not be ':', because an exception will be raised. This property has no effect if you use the version server lower than Oracle 9.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.4 ConnectionTimeout Property

Used to specify the time to wait for a connection to open before raising an exception.

Class

TOraSessionOptions

Syntax

```
property ConnectionTimeout: integer default
DefValOraConnectionTimeout;
```

Remarks

Use the ConnectionTimeout property to specify the time to wait for a connection to open before raising an exception. Works only when Direct mode is set to True.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.5 ConvertEOL Property

Affects the line break behavior in string fields and parameters.

Class

TOraSessionOptions

Syntax

```
property ConvertEOL: boolean default False;
```

Remarks

Affects the line break behavior in string fields and parameters. When fetching strings (including CLOBs and LONGs) with ConvertEOL = True dataset converts their line breaks from LF to CRLF form. And when posting strings to server with ConvertEOL turned on their line breaks converted from CRLF to LF form. By default, strings are not converted.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.6 DateFormat Property

Used to specify the default date format used when Oracle makes conversions from internal date format into string values and vice versa.

Class

TOraSessionOptions

Syntax

```
property DateFormat: string stored FIsDateFormatStored;
```

Remarks

Use the DateFormat property to specify the default date format used when Oracle makes conversions from internal date format into string values and vice versa. An example of valid expression for this property could be "MM/DD/YYYY".

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.7 DateLanguage Property

Used to specify the default language used when Oracle parses internal date format into string values and vice versa.

Class

TOraSessionOptions

Syntax

```
property DateLanguage: string stored FIsDateLanguageStored;
```

Remarks

Use the DateLanguage property to specify the default language used when Oracle parses internal date format into string values and vice versa. Examples of valid expressions for this property could be "French", "German" etc.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.8 Direct Property

Used for ODAC to connect directly over TCP/IP (in Direct mode) and without requiring Oracle software on the client side.

Class

TOraSessionOptions

Syntax

```
property Direct: boolean default DefValDirect;
```

Remarks

If the Direct property is set to True, ODAC connects directly over TCP/IP (in Direct mode) and does not require Oracle software on the client side. Otherwise, ODAC connects in Client mode. Supported by ODAC Professional and Professional with source code editions.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.9 EnableBCD Property

Used to enable currency type. Default value of this option is False.

Class

TOraSessionOptions

Syntax

```
property EnableBCD: boolean;
```

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.10 EnableFMTBCD Property

Used to enable using FMTBCD instead of float for large integer numbers to keep precision.

Class

TOraSessionOptions

Syntax

```
property EnableFMTBCD: boolean;
```

Remarks

Use the EnableFMTBCD property to enable using FMTBCD instead of float for large integer numbers to keep precision.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.11 EnableIntegers Property

Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.

Class

TOraSessionOptions

Syntax

```
property EnableIntegers: boolean default True;
```

Remarks

When the EnableIntegers property is set to True ODAC maps Oracle numbers with precision less than 10 to TIntegerField. If EnableIntegers is set to False numbers are mapped to TFloatField or TOraNumberField.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.12 EnableLargeint Property

Used for ODAC to map Oracle numbers with precision less than 10 to TIntegerField.

Class

TOraSessionOptions

Syntax

```
property EnableLargeint: boolean default False;
```

Remarks

When the EnableIntegers property is set to True, ODAC maps Oracle numbers with precision less than 10 to TIntegerField. If EnableIntegers is set to False, numbers are mapped to TFloatField or TOraNumberField.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.13 EnableNumbers Property

Used for ODAC to map Oracle numbers with precision larger than 15 to TOraNumberField.

Class

TOraSessionOptions

Syntax

```
property EnableNumbers: boolean default False;
```

Remarks

When the EnableNumbers property is set to True ODAC maps Oracle numbers with precision larger than 15 to TOraNumberField. Otherwise they are mapped to TFloatFiled.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.14 EnableOraTimestamp Property

Used to create TOraTimeStampField for columns of TIMESTAMP data type.

Class

TOraSessionOptions

Syntax

```
property EnableOraTimestamp: boolean default True;
```

Remarks

When the EnableOraTimestamp property is set to True, TOraTimeStampField is created for columns of TIMESTAMP data type.

When False, standard TSQLTimeStampField is created for columns of TIMESTAMP data type.

5.14.1.23.2.15 IPVersion Property

Used to specify Internet Protocol Version.

Class

TOraSessionOptions

Syntax

```
property IPVersion: TIPVersion default DefValIPVersion;
```

Remarks

Use the IPVersion property to specify Internet Protocol Version.

Supported values:

- ivIPBoth - specifies that either Internet Protocol Version 6 (IPv6) or Version 4 (IPv4) will be used;
- ivIPv4 (default) - specifies that Internet Protocol Version 4 (IPv4) will be used;

- ivIPv6 - specifies that Internet Protocol Version 6 (IPv6) will be used.

Note : When the TIPVersion property is set to ivIPBoth, a connection attempt will be made via IPv6 if it is enabled on the operating system. If the connection attempt fails, a new connection attempt will be made via IPv4.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.16 OptimizerMode Property

Used to get or set the default optimizer mode for connection.

Class

TOraSessionOptions

Syntax

```
property OptimizerMode: TOptimizerMode default omDefault;
```

Remarks

Use the OptimizerMode property to get or set the default optimizer mode for connection.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.17 StatementCache Property

Used for ODAC to cache statement handles.

Class

TOraSessionOptions

Syntax

```
property StatementCache: boolean default False;
```

Remarks

When the StatementCache property is set to True, ODAC caches statement handles.

 Request Support DAC Forum Provide Feedback

5.14.1.23.2.18 StatementCacheSize Property

Used to specify the statement handle cache size.

Class

TOraSessionOptions

Syntax

```
property StatementCacheSize: integer default 20;
```

Remarks

Use the StatementCacheSize property to specify the statement handle cache size.

Note: If StatementCache property is set, you can use TOraDataSet.Options and TOraSQL.StatementCache to adjust performance of dataset and SQL components using this session.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.19 SubscriptionPort Property

Sets the client port used to receive notifications.

Class

TOraSessionOptions

Syntax

```
property SubscriptionPort: Integer default 0;
```

Remarks

Use the SubscriptionPort property to set the port number for receiving notifications (for example, for clients on a computer behind a firewall). This option applies only to OCI clients when there is an exact correspondence between the client and the server versions.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.20 UnicodeEnvironment Property

Enables or disables using OCI Unicode Environment.

Class

TOraSessionOptions

Syntax

```
property UnicodeEnvironment: boolean default False;
```

Remarks

When this option is enabled, Unicode characters can be used in SQL statements. Disable this option if you have encountered some problems with Unicode Environment.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.21 UseOCI7 Property

Used to force TOraSession use OCI 7 call style only.

Class

TOraSessionOptions

Syntax

```
property UseOCI7: boolean default False;
```

Remarks

Use the UseOCI7 property to force TOraSession use OCI 7 call style only.

Request Support DAC Forum Provide Feedback

5.14.1.23.2.22 UseUnicode Property

Used to enable or disable Unicode support.

Class

TOraSessionOptions

Syntax

```
property UseUnicode: boolean default DefValUseUnicode;
```

Remarks

Use the UseUnicode property to enable or disable Unicode support. Affects character data fetched from the server. When set to True all character data stored as WideStrings and TStringField is replaced with TWideStringFiled. Supported starting with Oracle 8.

Request Support DAC Forum Provide Feedback

5.14.1.24 TOraSQL Class

A component for executing SQL statements and calling stored procedures on the database server.

For a list of all members of this type, see TOraSQL members.

Unit

Ora

Syntax

```
TOraSQL = class(TCustomDASQL);
```

Remarks

The TOraSQL component is a direct descendant of the TCustomDASQL class.

Use The TOraSQL component when a client application must execute SQL statement or the PL/SQL block, and call stored procedure on the database server. The SQL statement should not retrieve rows from the database.

Inheritance Hierarchy

TCustomDASQL

TOraSQL

See Also

- TOraQuery
- TOraScript

Request Support DAC Forum Provide Feedback

5.14.1.24.1 Members

TOraSQL class overview.

Properties

Name	Description
ArrayLength	Used to set the Length propery to all parameters.
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
CommandTimeout	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Connection (inherited from **TCustomDASQL**)	Used to specify a connection object to use to connect to a data store.
Debug (inherited from **TCustomDASQL**)	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL (inherited from **TCustomDASQL**)	Used to return a SQL statement with expanded macros.
MacroCount (inherited from **TCustomDASQL**)	Used to get the number of macros associated with the

	Macros property.
Macros (inherited from **TCustomDASQL**)	Makes it possible to change SQL queries easily.
NonBlocking	Used to execute a SQL statement by a separate thread.
ParamCheck (inherited from **TCustomDASQL**)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount (inherited from **TCustomDASQL**)	Indicates the number of parameters in the Params property.
Params	Contains the parameters for a SQL statement.
ParamValues (inherited from **TCustomDASQL**)	Used to get or set the values of individual field parameters that are identified by name.
Prepared (inherited from **TCustomDASQL**)	Used to indicate whether a query is prepared for execution.
RowsAffected (inherited from **TCustomDASQL**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed	Used to return the number of rows processed by a SQL statement.
Session	Used to define the session to execute SQL in.
SQL (inherited from **TCustomDASQL**)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
SQLType	Used to get the typecode of the SQL statement being processed by an Oracle database server.
StatementCache	Used to get a value indicating whether Oracle

	resources assosiated with the current statement will be cached inside a session.
TemporaryLobUpdate	Specifies whether to use temporary LOBs for writing input and input/output LOB parameters into database when executing SQL statements.

Methods

Name	Description
BreakExec (inherited from **TCustomDASQL**)	Breaks execution of an SQL satatement on the server.
CreateProcCall	Assigns a PL/SQL block that calls stored procedure
ErrorOffset	Obtains zero-based starting byte position of a parse error detected by an Oracle server while processing SQL statement.
Execute (inherited from **TCustomDASQL**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDASQL**)	Checks whether TCustomDASQL still executes a SQL statement.
FindMacro (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
FindParam	Searches for and returns a parameter with the specified name.
MacroByName (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
ParamByName	Searches for and returns a parameter with the specified name.
Prepare (inherited from **TCustomDASQL**)	Allocates, opens, and parses cursor for a query.
UnPrepare (inherited from **TCustomDASQL**)	Frees the resources allocated for a previously prepared query on the server and client sides.

WaitExecuting (inherited from **TCustomDASQL**)	Waits until TCustomDASQL executes a SQL statement.

Events

Name	Description
AfterExecute (inherited from **TCustomDASQL**)	Occurs after a SQL statement has been executed.

Request Support DAC Forum Provide Feedback

5.14.1.24.2 Properties

Properties of the **TOraSQL** class.

For a complete list of the **TOraSQL** class members, see the TOraSQL Members topic.

Public

Name	Description
ArrayLength	Used to set the Length propery to all parameters.
ChangeCursor (inherited from **TCustomDASQL**)	Enables or disables changing screen cursor when executing commands in the NonBlocking mode.
Connection (inherited from **TCustomDASQL**)	Used to specify a connection object to use to connect to a data store.
Debug (inherited from **TCustomDASQL**)	Used to display the statement that is being executed and the values and types of its parameters.
FinalSQL (inherited from **TCustomDASQL**)	Used to return a SQL statement with expanded macros.
MacroCount (inherited from **TCustomDASQL**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDASQL**)	Makes it possible to change SQL queries easily.

ParamCheck (inherited from **TCustomDASQL**)	Used to specify whether parameters for the Params property are implicitly generated when the SQL property is being changed.
ParamCount (inherited from **TCustomDASQL**)	Indicates the number of parameters in the Params property.
ParamValues (inherited from **TCustomDASQL**)	Used to get or set the values of individual field parameters that are identified by name.
Prepared (inherited from **TCustomDASQL**)	Used to indicate whether a query is prepared for execution.
RowsAffected (inherited from **TCustomDASQL**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed	Used to return the number of rows processed by a SQL statement.
SQL (inherited from **TCustomDASQL**)	Used to provide a SQL statement that a TCustomDASQL component executes when the Execute method is called.
SQLType	Used to get the typecode of the SQL statement being processed by an Oracle database server.

Published

Name	Description
CommandTimeout	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
NonBlocking	Used to execute a SQL statement by a separate thread.

Params	Contains the parameters for a SQL statement.
Session	Used to define the session to execute SQL in.
StatementCache	Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.
TemporaryLobUpdate	Specifies whether to use temporary LOBs for writing input and input/output LOB parameters into database when executing SQL statements.

See Also

- TOraSQL Class
- TOraSQL Class Members

5.14.1.24.2.1 ArrayLength Property

Used to set the Length propery to all parameters.

Class

TOraSQL

Syntax

```
property ArrayLength: integer;
```

Remarks

Use the ArrayLength property to set the Length propery to all parameters. It is useful for DML array operations.

See Also

- TOraParam.Length

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.2 CommandTimeout Property

Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.

Class

TOraSQL

Syntax

```
property CommandTimeout: integer default 0;
```

Remarks

The wait time is specified in seconds to wait for the command to execute. The default value is 0. The value of 0 indicates there are no time limits (an attempt to execute a command will wait indefinitely).

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.3 NonBlocking Property

Used to execute a SQL statement by a separate thread.

Class

TOraSQL

Syntax

```
property NonBlocking: boolean;
```

Remarks

Set the NonBlocking property to True to execute a SQL statement by a separate thread.

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.4 Params Property

Contains the parameters for a SQL statement.

Class

TOraSQL

Syntax

```
property Params: TOraParams stored False;
```

Remarks

Contains the parameters for a SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). Params is a zero-based array of parameter records. Index specifies the array element to access.

An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName.

Example

Setting parameters in runtime.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with OraSQL do
begin
  SQL.Clear;
  SQL.Add('INSERT INTO Temp_Table(Id, Name)');
  SQL.Add('VALUES (:id, :Name)');
  ParamByName('Id').AsInteger := 55;
  Params[1].AsString := ' Green';
  Execute;
end;
end;
```

See Also

- TOraParam
- FindParam

5.14.1.24.2.5 RowsProcessed Property

Used to return the number of rows processed by a SQL statement.

Class

TOraSQL

Syntax

```
property RowsProcessed: integer;
```

Remarks

Use the RowsProcessed property to return the number of rows processed by a SQL statement. Useful for inserting, updating and deleting statements.

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.6 Session Property

Used to define the session to execute SQL in.

Class

TOraSQL

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session in which SQL will be executed. If Session is not connected, the Execute method calls Session.Connect.

See Also

- TOraSession

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.7 SQLType Property

Used to get the typecode of the SQL statement being processed by an Oracle database server.

Class

TOraSQL

Syntax

```
property SQLType: integer;
```

Remarks

Use the SQLType property to get the typecode of the SQL statement being processed by an Oracle database server.

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.8 StatementCache Property

Used to get a value indicating whether Oracle resources assosiated with the current statement will be cached inside a session.

Class

TOraSQL

Syntax

```
property StatementCache: boolean default False;
```

Remarks

OCI statement cache is enabled when you set TOraSessionOptions.StatementCacheSize in TOraSession.Options to a positive value. Set TOraSessionOptions.StatementCacheSize to 0 (default) or TOraSession.Options.StatemantCache to false if you don't want the statements to be cached.

 Request Support DAC Forum Provide Feedback

5.14.1.24.2.9 TemporaryLobUpdate Property

Specifies whether to use temporary LOBs for writing input and input/output LOB parameters into database when executing SQL statements.

Class

TOraSQL

Syntax

```
property TemporaryLobUpdate: boolean default False;
```

Remarks

If the TemporaryLobUpdate property is True, temporary LOBs are used to write input and input/output LOB parameters into database when executing SQL statements.

Request Support DAC Forum Provide Feedback

5.14.1.24.3 Methods

Methods of the **TOraSQL** class.

For a complete list of the **TOraSQL** class members, see the TOraSQL Members topic.

Public

Name	Description
BreakExec (inherited from **TCustomDASQL**)	Breaks execution of an SQL satatement on the server.
CreateProcCall	Assigns a PL/SQL block that calls stored procedure
ErrorOffset	Obtains zero-based starting byte position of a parse error detected by an Oracle server while processing SQL statement.
Execute (inherited from **TCustomDASQL**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDASQL**)	Checks whether TCustomDASQL still

	executes a SQL statement.
FindMacro (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
FindParam	Searches for and returns a parameter with the specified name.
MacroByName (inherited from **TCustomDASQL**)	Finds a macro with the specified name.
ParamByName	Searches for and returns a parameter with the specified name.
Prepare (inherited from **TCustomDASQL**)	Allocates, opens, and parses cursor for a query.
UnPrepare (inherited from **TCustomDASQL**)	Frees the resources allocated for a previously prepared query on the server and client sides.
WaitExecuting (inherited from **TCustomDASQL**)	Waits until TCustomDASQL executes a SQL statement.

See Also

- TOraSQL Class
- TOraSQL Class Members

Request Support DAC Forum Provide Feedback

5.14.1.24.3.1 CreateProcCall Method

Assigns a PL/SQL block that calls stored procedure

Class

TOraSQL

Syntax

```
procedure CreateProcCall(Name: string; Overload: integer = 0);
```

Parameters

Name

Holds the stored procedure name.

Overload
Holds the number of overloaded procedure.

Remarks

Call the CreateProcCall method to assign a PL/SQL block that calls stored procedure specified by Name to the SQL property. The Overload parameter must contain the number of overloaded procedure. Retrieves the information about parameters of the procedure from Oracle. After calling CreateProcCall you can execute a stored procedure by Execute method.

See Also

- TCustomDASQL.Execute
- TCustomDAConnection.ExecProc
- TOraStoredProc

 Request Support DAC Forum Provide Feedback

5.14.1.24.3.2 ErrorOffset Method

Obtains zero-based starting byte position of a parse error detected by an Oracle server while processing SQL statement.

Class

TOraSQL

Syntax

```
function ErrorOffset: integer;
```

Return Value
Holds the position of a parse error detected by an Oracle server while processing SQL statement.

Remarks

Call the ErrorOffset method to obtain zero-based starting byte position of a parse error detected by an Oracle server while processing SQL statement.

Note that statements which are longer than 64KB will have unpredictable effect on the ErrorOffset value. Refer to Programmer's Guide to the Oracle Call Interface for further

information.

 Request Support DAC Forum Provide Feedback

5.14.1.24.3.3 FindParam Method

Searches for and returns a parameter with the specified name.

Class

TOraSQL

Syntax

```
function FindParam(const Value: string): TOraParam;
```

Parameters

Value

Holds the stored procedure name.

Return Value

the parameter, if a match was found. Nil otherwise.

Remarks

Call the FindParam method to find a parameter with the name passed in the Name argument. If a match was found, FindParam returns the parameter. Otherwise, it returns nil.

See Also

- TOraParam
- ParamByName

 Request Support DAC Forum Provide Feedback

5.14.1.24.3.4 ParamByName Method

Searches for and returns a parameter with the specified name.

Class

TOraSQL

Syntax

```
function ParamByName(const Value: string): TOraParam;
```

Parameters

Value

Holds the parameter name.

Return Value

the parameter, if a match was found. Otherwise an exception is raised.

Remarks

Call the ParamByName method to find a parameter with the name passed in the Name argument.

If a match is found, ParamByName returns the parameter. Otherwise, an exception is raised.

Example

```
OraSQL1.Execute;
Edit1.Text := OraSQL1.ParamsByName('Contact').AsString;
```

See Also

- TOraParam
- FindParam

Request Support DAC Forum Provide Feedback

5.14.1.25 TOraStoredProc Class

A component for accessing and executing stored procedures and functions.

For a list of all members of this type, see TOraStoredProc members.

Unit

Ora

Syntax

```
TOraStoredProc = class(TCustomOraQuery);
```

Remarks

Use TOraStoredProc to access stored procedures on the database server.

You need only to define the StoredProcName property, and the SQL statement to call the stored procedure will be generated automatically.

Use the Execute method at runtime to generate request that instructs server to execute procedure and PrepareSQL to describe parameters at run time

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet
      TCustomOraQuery
        TOraStoredProc

See Also

- TOraQuery
- TOraSQL
- Updating Data with ODAC Dataset Components
- TCustomDAConnection.ExecProc

 Request Support DAC Forum Provide Feedback

5.14.1.25.1 Members

TOraStoredProc class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable

	the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database

	server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database

	server.
LockMode	Used to specify what kind of lock will be performed when editing a record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
Overload	Used to specify the overloading number.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.

Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.

SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StoredProcName	Used to specify the name of the stored procedure to call on the server.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
ExecProc	Executes a SQL statement on the server.

Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference

	shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the

	TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Describes the parameters of a stored procedure.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the

	dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached

	updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

 Request Support DAC Forum Provide Feedback

5.14.1.25.2 Properties

Properties of the **TOraStoredProc** class.

For a complete list of the **TOraStoredProc** class members, see the TOraStoredProc Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.

Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL

	statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by

	a separate thread.
Options (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.

SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional

	access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
LockMode	Used to specify what kind of lock will be performed when editing a record.
Overload	Used to specify the overloading number.
StoredProcName	Used to specify the name of the stored procedure to call on the server.

See Also

- TOraStoredProc Class
- TOraStoredProc Class Members

5.14.1.25.2.1 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TOraStoredProc

Syntax

```
property LockMode: TLockMode default lmNone;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

See Also

- TOraQuery.LockMode
- TOraTable.LockMode

Request Support DAC Forum Provide Feedback

5.14.1.25.2.2 Overload Property

Used to specify the overloading number.

Class

TOraStoredProc

Syntax

```
property Overload: integer default 0;
```

Remarks

Use the Overload property to specify the overloading number in case the procedure or function is a part of a package and is overloaded.

Request Support DAC Forum Provide Feedback

5.14.1.25.2.3 StoredProcName Property

Used to specify the name of the stored procedure to call on the server.

Class

TOraStoredProc

Syntax

```
property StoredProcName: string;
```

Remarks

Use the StoredProcName property to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

Request Support DAC Forum Provide Feedback

5.14.1.25.3 Methods

Methods of the **TOraStoredProc** class.

For a complete list of the **TOraStoredProc** class members, see the TOraStoredProc Members topic.

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.

CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
ExecProc	Executes a SQL statement on the server.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.

FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from	Provides a list of available key field names.

TCustomDADataSet)	
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.

Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Describes the parameters of a stored procedure.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously

	prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TOraStoredProc Class
- TOraStoredProc Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.25.3.1 ExecProc Method

Executes a SQL statement on the server.

Class

TOraStoredProc

Syntax

```
procedure ExecProc;
```

Remarks

The ExecProc method is the same as TCustomDADataSet.Execute method. It is included for compatibility with TStoredProc.

See Also

- TCustomDADataSet.Execute

 Request Support DAC Forum Provide Feedback

5.14.1.25.3.2 PrepareSQL Method

Describes the parameters of a stored procedure.

Class

TOraStoredProc

Syntax

```
procedure PrepareSQL;
```

Remarks

Use the PrepareSQL method to describe parameters of a stored procedure. If it is necessary, Execute method calls it automatically. You can define parameters at design time if ParametersEditor is opened.

Request Support DAC Forum Provide Feedback

5.14.1.26 TOraTimeStampField Class

A class providing access to the Oracle timestamp fields.

For a list of all members of this type, see TOraTimeStampField members.

Unit

Ora

Syntax

```
TOraTimeStampField = class(TField);
```

Remarks

TOraTimeStampField provides access to Oracle timestamp fields. The TOraTimeStampField class supports three data types: TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE. According to this, the TOraTimeStampField.DataType property has three valid values ftTimeStamp, ftTimeStampTZ and ftTimeStampLTZ.

You can access actual timestamp value using the AsDateTime, AsString and AsOraTimeStamp properties.

See Also

- TOraTimeStamp

 Request Support DAC Forum Provide Feedback

5.14.1.26.1 Members

TOraTimeStampField class overview.

Properties

Name	Description
AsTimeStamp	Used to provide access to a TOraTimeStamp object.
Format	Used to get or set the date-time format model for operations with the AsString property.

 Request Support DAC Forum Provide Feedback

5.14.1.26.2 Properties

Properties of the **TOraTimeStampField** class.

For a complete list of the **TOraTimeStampField** class members, see the TOraTimeStampField Members topic.

Public

Name	Description
AsTimeStamp	Used to provide access to a TOraTimeStamp object.

Published

Name	Description
Format	Used to get or set the date-time format model for

	operations with the AsString property.

See Also

- TOraTimeStampField Class
- TOraTimeStampField Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.26.2.1 AsTimeStamp Property

Used to provide access to a TOraTimeStamp object.

Class

TOraTimeStampField

Syntax

```
property AsTimeStamp: TOraTimeStamp;
```

Remarks

Use the AsTimeStamp property to get access to a TOraTimeStamp object which you can use for manipulations with timestamp value.

See Also

- TOraTimeStamp

 Request Support DAC Forum Provide Feedback

5.14.1.26.2.2 Format Property

Used to get or set the date-time format model for operations with the AsString property.

Class

TOraTimeStampField

Syntax

```
property Format: string;
```

Remarks

Use the Format property to get or set the date-time format model for operations with the AsString property. Format string should be an Oracle date-time string.

See Also

- TOraTimeStamp.Format

Request Support DAC Forum Provide Feedback

5.14.1.27 TOraTrace Class

A component allowing starting and stopping a SQL trace for a specified session. This component provides access to the DBMS_TRACE package.

For a list of all members of this type, see TOraTrace members.

Unit

Ora

Syntax

```
TOraTrace = class(TComponent);
```

Remarks

Use the TOraTrace component to start and stop a SQL trace for a session. The component also provides access to functionality of DBMS_TRACE package to control the PL/SQL trace.

SQL trace can be useful in performance diagnostics. It can help to determine in detail how applications/users access the database.

The TOraTrace component automatically starts the trace when its TOraSession component connects to a database or when you assign already connected session to the TOraTrace.Session property. The SQL trace is started if TOraTrace.SqlTraceMode <> []. The PL/SQL trace is started if TOraTrace.PlSqlTraceMode <> [].

5.14.1.27.1 Members

TOraTrace class overview.

Properties

Name	Description
Enabled	Used to enable the trace.
MaxTraceFileSize	Used to limit the size of the SQL trace dump file.
PlSqlTraceMode	Defines the level of PL/SQL trace.
Session	used to specify the session on which a trace will be enabled.
SqlTraceMode	Used to determine the level of SQL trace.
State	Used to define wheter SQL trace and PL/SQL trace are active.
TraceFileIdentifier	Used to add a string to the SQL trace dump file name.

Methods

Name	Description
GetSessionPID	Returns the operating system process identifier for a process that owns the current session.
GetTraceFileName	Returns the name of a dump file on the database server to which SQL trace data is written.
PlSqlTraceComment	Provides a comment on the PL/SQL trace.
PlSqlTraceLimit	Limits the amount of storage used in the database for the PL/SQL trace data.

PlSqlTracePause	makes a pause in PL/SQL tracing.
PlSqlTraceResume	Resumes PL/SQL trace.
PlSqlTraceRunNumber	Returns a run number for the current PL/SQL trace.
PlSqlTraceStart	Starts PL/SQL trace data collection.
PlSqlTraceStop	Stops the PL/SQL trace.
SqlTraceStart	Starts the SQL trace.
SqlTraceStop	Stops the SQL trace.

Request Support DAC Forum Provide Feedback

5.14.1.27.2 Properties

Properties of the **TOraTrace** class.

For a complete list of the **TOraTrace** class members, see the TOraTrace Members topic.

Public

Name	Description
State	Used to define wheter SQL trace and PL/SQL trace are active.

Published

Name	Description
Enabled	Used to enable the trace.
MaxTraceFileSize	Used to limit the size of the SQL trace dump file.
PlSqlTraceMode	Defines the level of PL/SQL trace.
Session	used to specify the session on which a trace will be enabled.
SqlTraceMode	Used to determine the level of SQL trace.

TraceFileIdentifier	Used to add a string to the SQL trace dump file name.

See Also

- TOraTrace Class
- TOraTrace Class Members

 Request Support DAC Forum Provide Feedback

5.14.1.27.2.1 Enabled Property

Used to enable the trace.

Class

TOraTrace

Syntax

```
property Enabled: boolean default True;
```

Remarks

Use the Enabled property to enable or disable the trace. Set Enabled to False to disable the trace.

See Also

- SqlTraceStart
- SqlTraceStop

 Request Support DAC Forum Provide Feedback

5.14.1.27.2.2 MaxTraceFileSize Property

Used to limit the size of the SQL trace dump file.

Class

TOraTrace

Syntax

```
property MaxTraceFileSize: integer default
DEFAULT_TRACE_FILE_SIZE;
```

Remarks

Use the MaxTraceFileSize property to limit the size of the SQL trace dump file.

 Request Support DAC Forum Provide Feedback

5.14.1.27.2.3 PlSqlTraceMode Property

Defines the level of PL/SQL trace.

Class

TOraTrace

Syntax

```
property PlSqlTraceMode: TPlSqlTraceMode default [];
```

Remarks

Use the PlSqlTraceMode property to define the level of PL/SQL trace.

Note: PL/SQL program unit is enabled when it is compiled with debug information.

 Request Support DAC Forum Provide Feedback

5.14.1.27.2.4 Session Property

used to specify the session on which a trace will be enabled.

Class

TOraTrace

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session on which a trace will be enabled.

Request Support DAC Forum Provide Feedback

5.14.1.27.2.5 SqlTraceMode Property

Used to determine the level of SQL trace.

Class

TOraTrace

Syntax

```
property SqlTraceMode: TSqlTraceMode default
[smTypicalStatistics, smTimedStatistics];
```

Remarks

Use the SqlTraceMode property to define the level of SQL trace.

Request Support DAC Forum Provide Feedback

5.14.1.27.2.6 State Property

Used to define wheter SQL trace and PL/SQL trace are active.

Class

TOraTrace

Syntax

```
property State: TTraceState;
```

Remarks

Use the State property to detect whether SQL trace and PL/SQL trace are active.

Request Support DAC Forum Provide Feedback

Reserved.

5.14.1.27.2.7 TraceFileIdentifier Property

Used to add a string to the SQL trace dump file name.

Class

TOraTrace

Syntax

```
property TraceFileIdentifier: string;
```

Remarks

Use the TraceFileIdentifier property to add a string to the name of the SQL trace dump file. If you set TraceFileIdentifier property to some non-empty string then this string will be added to the name of SQL trace dump file. It makes finding of the SQL trace dump file easier.

See Also

- GetTraceFileName
- GetSessionPID

Request Support DAC Forum Provide Feedback

5.14.1.27.3 Methods

Methods of the **TOraTrace** class.

For a complete list of the **TOraTrace** class members, see the TOraTrace Members topic.

Public

Name	Description
GetSessionPID	Returns the operating system process identifier for a process that owns the current session.
GetTraceFileName	Returns the name of a dump file on the database server to which SQL trace data is

	written.
PlSqlTraceComment	Provides a comment on the PL/SQL trace.
PlSqlTraceLimit	Limits the amount of storage used in the database for the PL/SQL trace data.
PlSqlTracePause	makes a pause in PL/SQL tracing.
PlSqlTraceResume	Resumes PL/SQL trace.
PlSqlTraceRunNumber	Returns a run number for the current PL/SQL trace.
PlSqlTraceStart	Starts PL/SQL trace data collection.
PlSqlTraceStop	Stops the PL/SQL trace.
SqlTraceStart	Starts the SQL trace.
SqlTraceStop	Stops the SQL trace.

See Also

- TOraTrace Class
- TOraTrace Class Members

Request Support DAC Forum Provide Feedback

5.14.1.27.3.1 GetSessionPID Method

Returns the operating system process identifier for a process that owns the current session.

Class

TOraTrace

Syntax

```
function GetSessionPID: integer;
```

Return Value

the operating system process identifier for a process that owns the current session.

Remarks

Call the GetSessionPID method to return the operating system process identifier for a process that owns the current session. This identifier can be usefull to find the dump file in which SQL trace data is written. The name of the dump file contains this identifier.

See Also

- GetTraceFileName
- TraceFileIdentifier

Request Support DAC Forum Provide Feedback

5.14.1.27.3.2 GetTraceFileName Method

Returns the name of a dump file on the database server to which SQL trace data is written.

Class

TOraTrace

Syntax

```
function GetTraceFileName: string;
```

Return Value

the name of a dump file on the database server to which SQL trace data is written.

Remarks

Call the GetTraceFileName method to return the name of a dump file on the database server to which SQL trace data is written. The file name is returned with the path to the file.

Note: In some versions of Oracle database the dump file name format can be different from the one returned by the GetTraceFileName method.

See Also

- GetSessionPID
- TraceFileIdentifier

Request Support DAC Forum Provide Feedback

5.14.1.27.3.3 PlSqlTraceComment Method

Provides a comment on the PL/SQL trace.

Class

TOraTrace

Syntax

```
procedure PlSqlTraceComment(const Comment: string);
```

Parameters

Comment

Holds the comment for the PL/SQL trace.

Remarks

Call the PlSqlTraceComment method to set a comment on the PL/SQL trace.

Request Support DAC Forum Provide Feedback

5.14.1.27.3.4 PlSqlTraceLimit Method

Limits the amount of storage used in the database for the PL/SQL trace data.

Class

TOraTrace

Syntax

```
procedure PlSqlTraceLimit(Limit: integer = 8192);
```

Parameters

Limit

Holds the limit size for the storage used for the PL/SQL trace data.

Remarks

Call the PlSqlTraceLimit method to limit the amount of storage used in the database for the PL/SQL trace data.

Request Support DAC Forum Provide Feedback

Reserved.

5.14.1.27.3.5 PlSqlTracePause Method

makes a pause in PL/SQL tracing.

Class

TOraTrace

Syntax

```
procedure PlSqlTracePause;
```

Remarks

Call the PlSqlTracePause method to pause PL/SQL trace.

See Also

- PlSqlTraceResume

Request Support DAC Forum Provide Feedback

5.14.1.27.3.6 PlSqlTraceResume Method

Resumes PL/SQL trace.

Class

TOraTrace

Syntax

```
procedure PlSqlTraceResume;
```

Remarks

Call the PlSqlTraceResume method to resume PL/SQL trace.

See Also

- PlSqlTracePause

Request Support DAC Forum Provide Feedback

5.14.1.27.3.7 PlSqlTraceRunNumber Method

Returns a run number for the current PL/SQL trace.

Class

TOraTrace

Syntax

```
function PlSqlTraceRunNumber: integer;
```

Return Value

a run number for the current PL/SQL trace.

Remarks

Call the PlSqlTraceRunNumber method to return a run number for the current PL/SQL trace. This number can be used to retrieve information from the trace tables.

Request Support DAC Forum Provide Feedback

5.14.1.27.3.8 PlSqlTraceStart Method

Starts PL/SQL trace data collection.

Class

TOraTrace

Syntax

```
procedure PlSqlTraceStart;
```

Remarks

When Enabled property is False, call the PlSqlTraceStart method to start PL/SQL trace data collection. Setting Enabled property to True is another way to start the PL/SQL trace.

See Also

- Enabled
- PlSqlTraceStop
- PlSqlTracePause
- PlSqlTraceResume

 Request Support DAC Forum Provide Feedback

5.14.1.27.3.9 PlSqlTraceStop Method

Stops the PL/SQL trace.

Class

TOraTrace

Syntax

```
procedure PlSqlTraceStop;
```

Remarks

Call the PlSqlTraceStop method to stop the PL/SQL trace. Setting the Enabled property to False is another way to stop the PL/SQL trace.

See Also

- Enabled
- PlSqlTraceStart
- PlSqlTracePause
- PlSqlTraceResume

 Request Support DAC Forum Provide Feedback

5.14.1.27.3.10 SqlTraceStart Method

Starts the SQL trace.

Class

TOraTrace

Syntax

```
procedure SqlTraceStart;
```

Remarks

When the Enabled property is False, call the SqlTraceStart method to start the SQL trace. Setting the Enabled property to True is another way to start the SQL trace.

See Also

- Enabled
- SqlTraceStop

 Request Support DAC Forum Provide Feedback

5.14.1.27.3.11 SqlTraceStop Method

Stops the SQL trace.

Class

TOraTrace

Syntax

```
procedure SqlTraceStop;
```

Remarks

Call the SqlTraceStop method to stop the SQL trace. Setting the Enabled property to False is another way to stop the SQL trace.

See Also

- Enabled
- SqlTraceStart

 Request Support DAC Forum Provide Feedback

5.14.1.28 TOraUpdateSQL Class

A component for tuning update operations for the DataSet component.

For a list of all members of this type, see TOraUpdateSQL members.

Unit

Ora

Syntax

```
TOraUpdateSQL = class(TCustomDAUpdateSQL);
```

Remarks

Use the TOraUpdateSQL component to provide DML statements for the dataset components that return read-only result set. This component also allows setting objects that can be used for executing update operations. You may prefer to use directly SQLInsert, SQLUpdate, and SQLDelete properties of the TCustomDADataSet descendants.

Inheritance Hierarchy

TCustomDAUpdateSQL
  TOraUpdateSQL

See Also

- TOraDataSet.UpdateObject

5.14.1.28.1 Members

TOraUpdateSQL class overview.

Properties

Name	Description
DataSet (inherited from **TCustomDAUpdateSQL**)	Used to hold a reference to the TCustomDADataSet object that is being updated.

DeleteObject (inherited from **TCustomDAUpdateSQL**)	Provides ability to perform advanced adjustment of the delete operations.
DeleteSQL (inherited from **TCustomDAUpdateSQL**)	Used when deleting a record.
InsertObject (inherited from **TCustomDAUpdateSQL**)	Provides ability to perform advanced adjustment of insert operations.
InsertSQL (inherited from **TCustomDAUpdateSQL**)	Used when inserting a record.
LockObject (inherited from **TCustomDAUpdateSQL**)	Provides ability to perform advanced adjustment of lock operations.
LockSQL (inherited from **TCustomDAUpdateSQL**)	Used to lock the current record.
ModifyObject (inherited from **TCustomDAUpdateSQL**)	Provides ability to perform advanced adjustment of modify operations.
ModifySQL (inherited from **TCustomDAUpdateSQL**)	Used when updating a record.
RefreshObject (inherited from **TCustomDAUpdateSQL**)	Provides ability to perform advanced adjustment of refresh operations.
RefreshSQL (inherited from **TCustomDAUpdateSQL**)	Used to specify an SQL statement that will be used for refreshing the current record by TCustomDADataSet.RefreshRecord procedure.
SQL (inherited from **TCustomDAUpdateSQL**)	Used to return a SQL statement for one of the ModifySQL, InsertSQL, or DeleteSQL properties.

Methods

Name	Description
Apply (inherited from **TCustomDAUpdateSQL**)	Sets parameters for a SQL statement and executes it to update a record.
ExecSQL (inherited from **TCustomDAUpdateSQL**)	Executes a SQL statement.

Reserved.

5.14.1.29 TOraXMLField Class

A class providing access to the Oracle SYS.XMLTYPE objects.

For a list of all members of this type, see TOraXMLField members.

Unit

Ora

Syntax

```
TOraXMLField = class(TField);
```

Remarks

TOraXMLField provides access to the Oracle SYS.XMLTYPE objects.

The TMSXMLField.DataType property values equal to ftXML. You can access actual XML document using the AsString and TOraXMLField.AsXML properties.

See Also

- TOraXML

5.14.1.29.1 Members

TOraXMLField class overview.

Properties

Name	Description
AsXML	Used to provide access to a TOraXML object.

5.14.1.29.2 Properties

Properties of the **TOraXMLField** class.

For a complete list of the **TOraXMLField** class members, see the TOraXMLField Members topic.

Public

Name	Description
AsXML	Used to provide access to a TOraXML object.

See Also

- TOraXMLField Class
- TOraXMLField Class Members

Request Support DAC Forum Provide Feedback

5.14.1.29.2.1 AsXML Property

Used to provide access to a TOraXML object.

Class

TOraXMLField

Syntax

```
property AsXML: TOraXML;
```

Remarks

Use the AsXML property to get access to TOraXML object that can be used for manipulations with an XML document.

See Also

- TOraXML

Request Support DAC Forum Provide Feedback

Reserved.

5.14.2 Types

Types in the **Ora** unit.

Types

Name	Description
TConnectChangeEvent	This type is used for the TOraSession.OnConnectChange event.
TFailoverEvent	This Type is used for the TOraSession.OnFailover event.
TOraChangeNotificationEvent	This type is used for the TOraChangeNotification.OnChange event.
TPlSqlTraceMode	Specifies the level of PL/SQL trace.
TSqlTraceMode	Specifies the level of SQL trace statistics level.

Request Support DAC Forum Provide Feedback

5.14.2.1 TConnectChangeEvent Procedure Reference

This type is used for the TOraSession.OnConnectChange event.

Unit

Ora

Syntax

```
TConnectChangeEvent = procedure (Sender: TObject; Connected:
boolean) of object;
```

Parameters

Sender
An object that raised the event.

Connected
True, if connection is active.

 Request Support DAC Forum Provide Feedback

5.14.2.2 TFailoverEvent Procedure Reference

This Type is used for the TOraSession.OnFailover event.

Unit

Ora

Syntax

```
TFailoverEvent = procedure (Sender: TObject; FailoverState:
TFailoverState; FailoverType: TFailoverType; var Retry: boolean)
of object;
```

Parameters

Sender
An object that raised the event.

FailoverState
The failover state.

FailoverType
The type of failover.

Retry
True, if performing of the failover should be retried.

 Request Support DAC Forum Provide Feedback

5.14.2.3 TOraChangeNotificationEvent Procedure Reference

This type is used for the TOraChangeNotification.OnChange event.

Unit

Ora

Syntax

```
TOraChangeNotificationEvent = procedure (Sender: TObject;
NotifyType: TChangeNotifyEventType; TableChanges:
TNotifyTableChanges) of object;
```

Parameters

Sender
An object that raised the event.

NotifyType
The type of the event that occurred.

TableChanges
Holds the information on all table changes.

 Request Support DAC Forum Provide Feedback

5.14.2.4 TPlSqlTraceMode Set

Specifies the level of PL/SQL trace.

Unit

Ora

Syntax

```
TPlSqlTraceMode = set of ( pmAllCalls, pmEnabledCalls,
pmAllExceptions, pmEnabledExceptions, pmAllSql, pmEnabledSql,
pmAllLines, pmEnabledLines);
```

 Request Support DAC Forum Provide Feedback

5.14.2.5 TSqlTraceMode Set

Specifies the level of SQL trace statistics level.

Unit

Ora

Syntax

```
TSqlTraceMode = set of ( smBasicStatistics, smTypicalStatistics,
smAllStatistics, smBindVariables, smWaitEvents, smTimedStatistics);
```

 Request Support DAC Forum Provide Feedback

5.14.3 Enumerations

Enumerations in the **Ora** unit.

Enumerations

Name	Description
TFailoverState	Indicates the failover state.
TFailoverType	Specifies the failover type.
TOraIsolationLevel	Specifies the way the transactions containing database modifications are handled.
TRefreshMode	Defines when to refresh an editing record.
TSequenceMode	Specifies the method used internally to generate sequenced field.

Request Support DAC Forum Provide Feedback

5.14.3.1 TFailoverState Enumeration

Indicates the failover state.

Unit

Ora

Syntax

```
TFailoverState = (fsEnd, fsAbort, fsReauth, fsBegin, fsError);
```

Values

Value	Meaning
fsAbort	Indicates that failover was unsuccessful and there is no option of retrying.
fsBegin	Indicates that failover has detected a lost connection and failover is starting.
fsEnd	Indicates successful completion of failover.

fsError	Indicates that an error occurred while trying to re-establish the connection.
fsReauth	Indicates that a user handle has been re-authenticated.

Request Support DAC Forum Provide Feedback

5.14.3.2 TFailoverType Enumeration

Specifies the failover type.

Unit

Ora

Syntax

```
TFailoverType = (ftSession, ftSelect);
```

Values

Value	Meaning
ftSelect	Indicates that the user has requested select failover as well.
ftSession	Indicates that the user has requested only session failover.

Request Support DAC Forum Provide Feedback

5.14.3.3 TOraIsolationLevel Enumeration

Specifies the way the transactions containing database modifications are handled.

Unit

Ora

Syntax

```
TOraIsolationLevel = (ilReadCommitted, ilSerializable, ilReadOnly);
```

Values

Value	Meaning

ilReadCommitted	If the transaction contains DML that requires row locks held by another transaction, then the DML statement waits until the row locks are released. The default Oracle behavior.
ilReadOnly	All subsequent queries in that transaction only see changes committed before the transaction began. Read-only transactions are useful for reports that run multiple queries against one or more tables while other users update these same tables.
ilSerializable	Specifies serializable transaction isolation mode as defined in the SQL92 standard. If a serializable transaction contains data manipulation language (DML) that attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the DML statement fails.

Request Support DAC Forum Provide Feedback

5.14.3.4 TRefreshMode Enumeration

Defines when to refresh an editing record.

Unit

Ora

Syntax

```
TRefreshMode = (rmNone, rmAfterInsert, rmAfterUpdate, rmAlways);
```

Values

Value	Meaning
rmAfterInsert	Refresh is performed after inserting a record.
rmAfterUpdate	Refresh is performed after updating a record.
rmAlways	Refresh is performed after inserting and updating a record.
rmNone	No refresh is performed. The default value.

Request Support DAC Forum Provide Feedback

5.14.3.5 TSequenceMode Enumeration

Specifies the method used internally to generate sequenced field.

Unit

Ora

Syntax

```
TSequenceMode = (smInsert, smPost);
```

Values

Value	Meaning
smInsert	New record is inserted into the dataset where the first key field is populated with a sequenced value. An application may modify this field before posting the record to the database.
smPost	Database server populates the key field with a sequenced value when application posts the record to the database. Any value put into the key field before post will be overwritten.

5.14.4 Variables

Variables in the **Ora** unit.

Variables

Name	Description
DefSession	Read this variable to get pointer to default session object. Same as DefaultSession function.
OraQueryCompatibilityMode	All TOraQuery components in project become editable, and can be modified by the end users.
Sessions	Holds pointers to all TOraSession objects of an application.

UseDefSession	When set to True enables TOraDataSet and TOraSQL components to use default session if they are not attached to any session.

 Request Support DAC Forum Provide Feedback

5.14.4.1 DefSession Variable

Read this variable to get pointer to default session object. Same as DefaultSession function.

Unit

Ora

Syntax

```
DefSession: TOraSession;
```

 Request Support DAC Forum Provide Feedback

5.14.4.2 OraQueryCompatibilityMode Variable

All TOraQuery components in project become editable, and can be modified by the end users.

Unit

Ora

Syntax

```
OraQueryCompatibilityMode: boolean = False;
```

Remarks

Before ODAC 6, TOraQuery could be editable only when InsertSQL, UpdateSQL, and DeleteSQL properties are assigned. The ability to generate update SQL statements with TOraQuery automatically was added in ODAC 6.00.0.4. Therefore, after upgrading your ODAC to the sixth version, all TOraQuery components in you project become editable, and

can be modified by the end users. To restore the old behavior, set the OraQueryCompatibilityMode variable to True.

 Request Support DAC Forum Provide Feedback

5.14.4.3 Sessions Variable

Holds pointers to all TOraSession objects of an application.

Unit

Ora

Syntax

```
Sessions: TSessionList;
```

 Request Support DAC Forum Provide Feedback

5.14.4.4 UseDefSession Variable

When set to True enables TOraDataSet and TOraSQL components to use default session if they are not attached to any session.

Unit

Ora

Syntax

```
UseDefSession: boolean;
```

 Request Support DAC Forum Provide Feedback

5.15 OraAlerter

This unit contains implementation of the TOraAlerter component.

Classes

Name	Description
TOraAlerter	A component allowing transferring messages between sessions or client applications.

Types

Name	Description
TOnEventEvent	This type is used for the TOraAlerter.OnEvent event.
TOnTimeOutEvent	This type is used for the TOraAlerter.OnTimeOut event.

Enumerations

Name	Description
TEventType	Specifies the kind of messages used by the TOraAlerter component.

5.15.1 Classes

Classes in the **OraAlerter** unit.

Classes

Name	Description
TOraAlerter	A component allowing transferring messages between sessions or client applications.

5.15.1.1 TOraAlerter Class

A component allowing transferring messages between sessions or client applications.

For a list of all members of this type, see TOraAlerter members.

Unit

OraAlerter

Syntax

```
TOraAlerter = class(TDAAlerter);
```

Remarks

The TOraAlerter component allows to transfer messages between sessions or client applications. Use the EventType property to specify what kind of messages will be used. The TOraAlerter component supports Oracle alerts by DBMS_ALERT and pipes by DBMS_PIPE. You can send and get messages with the help of this component. When TOraAlerter is started by Start method it waits messages in a background thread. If a message is received, the OnEvent event occurs. If no messages were received during the TimeOut time, the OnTimeOut event occurs.

Note: Alerts are transaction-based. This means that the waiting session does not get alert until the transaction signaling the alert commits.

Inheritance Hierarchy

TDAAlerter

TOraAlerter

5.15.1.1.1 Members

TOraAlerter class overview.

Properties

Name	Description

Active (inherited from **TDAAlerter**)	Used to determine if TDAAlerter waits for messages.
AutoCommit	Used to specify whether the TOraAlerter object should commit transaction in its Session property after calling the SendEvent method to send a named message to an Oracle server.
AutoRegister (inherited from **TDAAlerter**)	Used to automatically register events whenever connection opens.
Connection (inherited from **TDAAlerter**)	Used to specify the connection for TDAAlerter.
Events	Used to set the names of waiting alerts or pipes.
EventType	Used to define the kind of messages used by the TOraAleter component.
Interval	Specifies the time after which TOraAlerter starts waiting process.
Session	Specifies the session for TOraAlerter to create an internal TOraSession object based on this session settings.
TimeOut	Used to set the time for the TOraAlerter component to wait for a message.

Methods

Name	Description
GetMessage	Overloaded.
NextItemType	Returns the datatype of the next message found in the received named pipe local buffer.
NextMessageType	Returns the datatype of the next message found in the

	received named pipe local buffer.
PackMessage	Places messages into the named pipe local buffer.
PurgePipe	Removes all incoming messages from the pipe's input buffer.
PutMessage	Places messages into the named pipe local buffer.
SendEvent (inherited from **TDAAlerter**)	Sends an event with Name and content Message.
SendMessage	Sends a named pipe event to other listening sessions.
SendPipeMessage	Sends a named pipe event to other listening sessions.
Start (inherited from **TDAAlerter**)	Starts waiting process.
Stop (inherited from **TDAAlerter**)	Stops waiting process.
UnpackMessage	Overloaded. Retrieves messages from a named pipe local buffer.

Events

Name	Description
OnError (inherited from **TDAAlerter**)	Occurs if an exception occurs in waiting process
OnEvent	Occurs if a message is received by waiting process.
OnTimeOut	Occurs if there were no messages during the TimeOut time.

Request Support DAC Forum Provide Feedback

5.15.1.1.2 Properties

Properties of the **TOraAlerter** class.

For a complete list of the **TOraAlerter** class members, see the TOraAlerter Members topic.

Public

Name	Description
Active (inherited from **TDAAlerter**)	Used to determine if TDAAlerter waits for messages.
AutoRegister (inherited from **TDAAlerter**)	Used to automatically register events whenever connection opens.
Connection (inherited from **TDAAlerter**)	Used to specify the connection for TDAAlerter.

Published

Name	Description
AutoCommit	Used to specify whether the TOraAlerter object should commit transaction in its Session property after calling the SendEvent method to send a named message to an Oracle server.
Events	Used to set the names of waiting alerts or pipes.
EventType	Used to define the kind of messages used by the TOraAleter component.
Interval	Specifies the time after which TOraAlerter starts waiting process.
Session	Specifies the session for TOraAlerter to create an internal TOraSession object based on this session settings.
TimeOut	Used to set the time for the TOraAlerter component to wait for a message.

See Also

- TOraAlerter Class
- TOraAlerter Class Members

5.15.1.1.2.1 AutoCommit Property

Used to specify whether the TOraAlerter object should commit transaction in its Session property after calling the SendEvent method to send a named message to an Oracle server.

Class

TOraAlerter

Syntax

```
property AutoCommit: boolean;
```

Remarks

Set the AutoCommit property to specify whether the TOraAlerter object should commit transaction in its Session property after calling the SendEvent method to send a named message to an Oracle server.

Setting AutoCommit to True instructs TOraAlerter to commit its session each time an event is sent. Otherwise listening session will have to wait till current transaction terminates and only then it will be notified by the server.

See Also

- TDAAlerter.SendEvent

5.15.1.1.2.2 Events Property

Used to set the names of waiting alerts or pipes.

Class

TOraAlerter

Syntax

```
property Events: string;
```

Remarks

Use the Events property to set the names of alerts or pipes which are waiting.

 Request Support DAC Forum Provide Feedback

5.15.1.1.2.3 EventType Property

Used to define the kind of messages used by the TOraAleter component.

Class

TOraAlerter

Syntax

```
property EventType: TEventType;
```

Remarks

Use EventType to specify what kind of messages is used by the TOraAleter component.

 Request Support DAC Forum Provide Feedback

5.15.1.1.2.4 Interval Property

Specifies the time after which TOraAlerter starts waiting process.

Class

TOraAlerter

Syntax

```
property Interval: integer default 0;
```

Remarks

If Interval property is greater than 0, TOraAlerter starts waiting process in Interval seconds after time out occurred.

5.15.1.1.2.5 Session Property

Specifies the session for TOraAlerter to create an internal TOraSession object based on this session settings.

Class

TOraAlerter

Syntax

```
property Session: TOraSession;
```

Remarks

Specifies the session for TOraAlerter to create an internal TOraSession object based on this session settings.

See Also

- TOraSession

5.15.1.1.2.6 TimeOut Property

Used to set the time for the TOraAlerter component to wait for a message.

Class

TOraAlerter

Syntax

```
property TimeOut: integer default 0;
```

Remarks

Use TimeOut property to set the time of waiting message by the TOraAlerter component.

5.15.1.1.3 Methods

Methods of the **TOraAlerter** class.

For a complete list of the **TOraAlerter** class members, see the TOraAlerter Members topic.

Public

Name	Description
GetMessage	Overloaded.
NextItemType	Returns the datatype of the next message found in the received named pipe local buffer.
NextMessageType	Returns the datatype of the next message found in the received named pipe local buffer.
PackMessage	Places messages into the named pipe local buffer.
PurgePipe	Removes all incoming messages from the pipe's input buffer.
PutMessage	Places messages into the named pipe local buffer.
SendEvent (inherited from **TDAAlerter**)	Sends an event with Name and content Message.
SendMessage	Sends a named pipe event to other listening sessions.
SendPipeMessage	Sends a named pipe event to other listening sessions.
Start (inherited from **TDAAlerter**)	Starts waiting process.
Stop (inherited from **TDAAlerter**)	Stops waiting process.
UnpackMessage	Overloaded. Retrieves messages from a named pipe local buffer.

See Also

- TOraAlerter Class
- TOraAlerter Class Members

 Request Support DAC Forum Provide Feedback

5.15.1.1.3.1 GetMessage Method

Class

TOraAlerter

Overload List

Name	Description
GetMessage	Retrieves messages from a named pipe local buffer.
GetMessage(**var** Item: variant)	Retrieves messages from a named pipe local buffer.

 Request Support DAC Forum Provide Feedback

Retrieves messages from a named pipe local buffer.

Class

TOraAlerter

Syntax

```
function GetMessage: variant; overload;
```

Return Value

True, if the Item parameter holds not Null variant value or False otherwise.

Remarks

Call the GetMessage method to retrieve messages from a named pipe local buffer.

GetMessage is desined to work only with Oracle DBMS_PIPE communication package which is the case only if TOraAlerter.EventType property has been set to etPipe.

Implemntation with the parameter will return True if the Item parameter holds not Null variant value or False otherwise.

Note: This method is considered obsolete now. In newer projects use functionally equivalent TOraAlerter.UnpackMessage method instead.

Retrieves messages from a named pipe local buffer.

Class

TOraAlerter

Syntax

```
function GetMessage(var Item: variant): variant; overload;
```

Parameters

Item

Holds a value received from the pipe.

Return Value

True, if the Item parameter holds not Null variant value or False otherwise.

5.15.1.1.3.2 NextItemType Method

Returns the datatype of the next message found in the received named pipe local buffer.

Class

TOraAlerter

Syntax

```
function NextItemType: TMessageType;
```

Return Value

the datatype of the next message found in the received named pipe local buffer.

Remarks

Use NextItemType to retrieve the datatype of the next message found in the received named pipe local buffer.

NextItemType is desined to work only with Oracle DBMS_PIPE communication package which is the case only if EventType property has been set to etPipe.

The mtNone return value indicates that no more messages are found in the local buffer.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.3 NextMessageType Method

Returns the datatype of the next message found in the received named pipe local buffer.

Class

TOraAlerter

Syntax

```
function NextMessageType: TMessageType;
```

Return Value

the datatype of the next message found in the received named pipe local buffer.

Remarks

Use NextMessageType to retrieve the datatype of the next message found in the received named pipe local buffer.

NextMessageType is desined to work only with Oracle DBMS_PIPE communication package which is the case only if EventType property has been set to etPipe.

The return value of mtNone indicates that no more messages are found in the local buffer.

Note: This method is considered obsolete now. In newer projects use functionally equivalent NextItemType method instead.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.4 PackMessage Method

Places messages into the named pipe local buffer.

Class

TOraAlerter

Syntax

```
procedure PackMessage(Item: variant);
```

Parameters

Item

Holds the value to be sent in a message.

Remarks

Call PackMessage to place messages into the named pipe local buffer. Local buffer is limited in size to 8192 bytes and besides the actual message values accommodates other internal data. Item will be dropped out if it doesn't fit into available free buffer space.

PackMessage is desined to work only with the Oracle DBMS_PIPE communication package which is the case only if the EventType property has been set to etPipe.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.5 PurgePipe Method

Removes all incoming messages from the pipe's input buffer.

Class

TOraAlerter

Syntax

```
procedure PurgePipe;
```

Remarks

Call PurgePipe to clear the pipe's input buffer.

PurgePipe is desined to work only with the Oracle DBMS_PIPE communication package. So

it can be called only if EventType property has been set to etPipe.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.6 PutMessage Method

Places messages into the named pipe local buffer.

Class

TOraAlerter

Syntax

```
procedure PutMessage(Item: variant);
```

Parameters

Item

Holds the value to be sent in a message.

Remarks

Call PutMessage to place messages into the named pipe local buffer. Local buffer is limited in size to 8192 bytes and besides the actual message values accommodates other internal data. Item will be dropped out if it doesn't fit into free buffer space.

PutMessage is desined to work only with the Oracle DBMS_PIPE communication package which is the case only if EventType property has been set to etPipe.

Note: This method is now considered obsolete. In newer projects use functionally equivalent PackMessage method instead.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.7 SendMessage Method

Sends a named pipe event to other listening sessions.

Class

TOraAlerter

Syntax

```
procedure SendMessage(Name: string = '');
```

Parameters

Name

Holds the name of the pipe.

Remarks

Use SendMessage procedure to send a named pipe event to other listening sessions. The event internally is a local buffer which has been previously filled in by the PutMessage method.

SendMessage is desined to work only with the Oracle DBMS_PIPE communication package which is the case only if EventType property has been set to etPipe.

Note: This method is considered obsolete now. In newer projects use functionally equivalent SendPipeMessage method instead.

Request Support DAC Forum Provide Feedback

5.15.1.1.3.8 SendPipeMessage Method

Sends a named pipe event to other listening sessions.

Class

TOraAlerter

Syntax

```
procedure SendPipeMessage(Name: string = '');
```

Parameters

Name

Holds the name of the pipe.

Remarks

Use SendPipeMessage procedure to send a named pipe event to other listening sessions. The event internally is a local buffer which has been previously filled in by the PackMessage

method.

SendPipeMessage is desined to work only with the Oracle DBMS_PIPE communication package which is the case only if EventType property has been set to etPipe.

 Request Support DAC Forum Provide Feedback

5.15.1.1.3.9 UnpackMessage Method

Retrieves messages from a named pipe local buffer.

Class

TOraAlerter

Overload List

Name	Description
UnpackMessage	Retrieves messages from a named pipe local buffer.
UnpackMessage(**var** Item: variant)	Retrieves messages from a named pipe local buffer.

 Request Support DAC Forum Provide Feedback

Retrieves messages from a named pipe local buffer.

Class

TOraAlerter

Syntax

```
function UnpackMessage: variant; overload;
```

Return Value

True, if the Item parameter holds not Null variant value or False otherwise.

Remarks

UnpackMessage function is used to retrieve messages from a named pipe local buffer.

UnpackMessage is desined to work only with the Oracle DBMS_PIPE communication package which is the case only if TOraAlerter.EventType property has been set to etPipe.

An implementation with the parameter will return True if the Item parameter holds not Null variant value or False otherwise.

 Request Support DAC Forum Provide Feedback

Retrieves messages from a named pipe local buffer.

Class

TOraAlerter

Syntax

```
function UnpackMessage(var Item: variant): variant; overload;
```

Parameters

Item

Holds a value received from the pipe.

Return Value

True, if the Item parameter holds not Null variant value or False otherwise.

 Request Support DAC Forum Provide Feedback

5.15.1.1.4 Events

Events of the **TOraAlerter** class.

For a complete list of the **TOraAlerter** class members, see the TOraAlerter Members topic.

Public

Name	Description
OnError (inherited from **TDAAlerter**)	Occurs if an exception occurs in waiting process

Published

Name	Description
OnEvent	Occurs if a message is received by waiting process.
OnTimeOut	Occurs if there were no messages during the TimeOut time.

See Also

- TOraAlerter Class
- TOraAlerter Class Members

 Request Support DAC Forum Provide Feedback

5.15.1.1.4.1 OnEvent Event

Occurs if a message is received by waiting process.

Class

TOraAlerter

Syntax

```
property OnEvent: TOnEventEvent;
```

Remarks

Occurs when waiting process receives some message. The event parameter is the name of an event (alert or pipe) and Message is its content.

See Also

- OnTimeOut

 Request Support DAC Forum Provide Feedback

5.15.1.1.4.2 OnTimeOut Event

Occurs if there were no messages during the TimeOut time.

Class

TOraAlerter

Syntax

```
property OnTimeOut: TOnTimeOutEvent;
```

Remarks

Occurs when there were no messages during the TimeOut time. Assign True to the Continue parameter to continue waiting messages. If Continue is False (by default) waiting process is stopped.

See Also

- TimeOut
- OnEvent

 Request Support DAC Forum Provide Feedback

5.15.2 Types

Types in the **OraAlerter** unit.

Types

Name	Description
TOnEventEvent	This type is used for the TOraAlerter.OnEvent event.
TOnTimeOutEvent	This type is used for the TOraAlerter.OnTimeOut event.

 Request Support DAC Forum Provide Feedback

5.15.2.1 TOnEventEvent Procedure Reference

This type is used for the TOraAlerter.OnEvent event.

Unit

OraAlerter

Syntax

```
TOnEventEvent = procedure (Sender: TObject; Event: string;
Message: string) of object;
```

Parameters

Sender
An object that raised the event.

Event
A name of event (alert or pipe).

Message
The content of message waiting process receives.

Request Support DAC Forum Provide Feedback

5.15.2.2 TOnTimeOutEvent Procedure Reference

This type is used for the TOraAlerter.OnTimeOut event.

Unit

OraAlerter

Syntax

```
TOnTimeOutEvent = procedure (Sender: TObject; var Continue:
boolean) of object;
```

Parameters

Sender
An object that raised the event.

Continue
True, if waiting messages process should be continued. If False (by default), the waiting process is stopped.

Request Support DAC Forum Provide Feedback

5.15.3 Enumerations

Enumerations in the **OraAlerter** unit.

Enumerations

Name	Description
TEventType	Specifies the kind of messages used by the TOraAlerter component.

Request Support DAC Forum Provide Feedback

5.15.3.1 TEventType Enumeration

Specifies the kind of messages used by the TOraAlerter component.

Unit

OraAlerter

Syntax

```
TEventType = (etAlert, etPipe);
```

Values

Value	Meaning
etAlert	Lets all listening sessions be notified when another session broadcasts its message with the corresponding name parameter. The default value.
etPipe	Causes only one of the listening sessions to receive a message.

Request Support DAC Forum Provide Feedback

5.16 OraAQ

This unit contains ODAC components for working with Oracle Advanced Queueing.

Classes

Name	Description
TDequeueOptions	A base class for setting default dequeue options for the queue or for using special dequeue options when calling the TOraQueue.Dequeue or TOraQueue.DequeueArray methods.
TEnqueueOptions	A base class for setting default or special enqueue options for a queue.
TOraQueue	A component providing access to Oracle Streams Advanced Queuing.
TOraQueueAdmin	A component for managing queues in a database.
TOraQueueTable	A component managing queue tables in a database.
TQueueAgent	A class reperesenting a producer or a consumer of a queue message.
TQueueAgents	A class holding a collection of the TQueueAgent objects.
TQueueMessage	A class representing a queue message.
TQueueMessageProperties	A class for setting or providing queue message properties.

Types

Name	Description
TQueueMessageEvent	This type is used for the TOraQueue.OnMessage event.

Enumerations

Name	Description
TDequeueMode	Specifies the locking behavior associated with the dequeue.
TQueueDeliveryMode	Specifies the type of the message that will be dequeued.
TQueueMessageGrouping	Specifies the message grouping behavior in the queues based on this table.
TQueueMessageState	Specifies the message state.
TQueueNavigation	Specifies the position of the message that will be retrieved.
TQueueSequenceDeviation	Specifies if a message should be enqueued before other messages.
TQueueSortList	Specifies the columnt that will be used as a sort key.
TQueueType	Specifies whether the queue being created is an exception queue or a normal queue.
TQueueVisibility	Specifies the transaction behaviour of the dequeue or enqueue request.

Request Support DAC Forum Provide Feedback

5.16.1 Classes

Classes in the **OraAQ** unit.

Classes

Name	Description
TDequeueOptions	A base class for setting default dequeue options for the queue or for using

	special dequeue options when calling the TOraQueue.Dequeue or TOraQueue.DequeueArray methods.
TEnqueueOptions	A base class for setting default or special enqueue options for a queue.
TOraQueue	A component providing access to Oracle Streams Advanced Queuing.
TOraQueueAdmin	A component for managing queues in a database.
TOraQueueTable	A component managing queue tables in a database.
TQueueAgent	A class reperesenting a producer or a consumer of a queue message.
TQueueAgents	A class holding a collection of the TQueueAgent objects.
TQueueMessage	A class representing a queue message.
TQueueMessageProperties	A class for setting or providing queue message properties.

Request Support DAC Forum Provide Feedback

5.16.1.1 TDequeueOptions Class

A base class for setting default dequeue options for the queue or for using special dequeue options when calling the TOraQueue.Dequeue or TOraQueue.DequeueArray methods.

For a list of all members of this type, see TDequeueOptions members.

Unit

OraAQ

Syntax

```
TDequeueOptions = class(TPersistent);
```

Remarks

Use the TDequeueOptions class for setting default dequeue options for the queue or for using special dequeue options when calling TOraQueue.Dequeue or TOraQueue.DequeueArray methods.

See Also

- TOraQueue.Dequeue
- TOraQueue.DequeueArray
- TOraQueue.DequeueOptions

5.16.1.1.1 Members

TDequeueOptions class overview.

Properties

Name	Description
ConsumerName	Used to specify the name of the consumer.
Correlation	Used to specify the correlation identifier of the message to be dequeued.
DeliveryMode	Used to specify the type of the message that will be dequeued.
DequeueCondition	Used to specify the conditional expression based on the message properties, the message data properties, and PL/SQL functions.
DequeueMode	Used to specify the locking behaviour associated with the dequeue.
MessageId	Used to specify the message ID for the message to be dequeued.

Navigation	Used to specify the position of the message that will be retrieved.
Transformation	Used to specify a transformation that will be applied after dequeuing the message.
Visibility	Used to specify the transactional behavior of the dequeue request.
WaitTimeout	Specifies the time to wait if there is no message matching the search criteria.

Request Support DAC Forum Provide Feedback

5.16.1.1.2 Properties

Properties of the **TDequeueOptions** class.

For a complete list of the **TDequeueOptions** class members, see the TDequeueOptions Members topic.

Public

Name	Description
MessageId	Used to specify the message ID for the message to be dequeued.

Published

Name	Description
ConsumerName	Used to specify the name of the consumer.
Correlation	Used to specify the correlation identifier of the message to be dequeued.
DeliveryMode	Used to specify the type of the message that will be dequeued.

DequeueCondition	Used to specify the conditional expression based on the message properties, the message data properties, and PL/SQL functions.
DequeueMode	Used to specify the locking behaviour associated with the dequeue.
Navigation	Used to specify the position of the message that will be retrieved.
Transformation	Used to specify a transformation that will be applied after dequeuing the message.
Visibility	Used to specify the transactional behavior of the dequeue request.
WaitTimeout	Specifies the time to wait if there is no message matching the search criteria.

See Also

- TDequeueOptions Class
- TDequeueOptions Class Members

Request Support DAC Forum Provide Feedback

5.16.1.1.2.1 ConsumerName Property

Used to specify the name of the consumer.

Class

TDequeueOptions

Syntax

```
property ConsumerName: string;
```

Remarks

Use ConsumerName property to specify the name of the consumer.

Request Support DAC Forum Provide Feedback

5.16.1.1.2.2 Correlation Property

Used to specify the correlation identifier of the message to be dequeued.

Class

TDequeueOptions

Syntax

```
property Correlation: string;
```

Remarks

Use Correlation property to specify the correlation identifier of the message to be dequeued.

Request Support DAC Forum Provide Feedback

5.16.1.1.2.3 DeliveryMode Property

Used to specify the type of the message that will be dequeued.

Class

TDequeueOptions

Syntax

```
property DeliveryMode: TQueueDeliveryMode default qdmPersistent;
```

Remarks

Use the DeliveryMode property to specify the type of the message that will be dequeued. Use it with Oracle 10 and higher.

See Also

- TEnqueueOptions.DeliveryMode

- TQueueMessageProperties.DeliveryMode

 Request Support DAC Forum Provide Feedback

5.16.1.1.2.4 DequeueCondition Property

Used to specify the conditional expression based on the message properties, the message data properties, and PL/SQL functions.

Class

TDequeueOptions

Syntax

```
property DequeueCondition: string;
```

Remarks

Use DequeueCondition property to specify the conditional expression based on the message properties, the message data properties, and PL/SQL functions. It should be a boolean expression like a SQL WHERE clause. DequeueCondition should not exceed 4000 characters.

 Request Support DAC Forum Provide Feedback

5.16.1.1.2.5 DequeueMode Property

Used to specify the locking behaviour associated with the dequeue.

Class

TDequeueOptions

Syntax

```
property DequeueMode: TDequeueMode default dqmRemove;
```

Remarks

Use DequeueMode property to specify the locking behavior associated with the dequeue.

5.16.1.1.2.6 MessageId Property

Used to specify the message ID for the message to be dequeued.

Class

TDequeueOptions

Syntax

```
property MessageId: string;
```

Remarks

Use the MessageId property to specify the message ID for the message to be dequeued.

5.16.1.1.2.7 Navigation Property

Used to specify the position of the message that will be retrieved.

Class

TDequeueOptions

Syntax

```
property Navigation: TQueueNavigation default qnNextMessage;
```

Remarks

Use the Navigation property to specify the position of the message that will be retrieved.

5.16.1.1.2.8 Transformation Property

Used to specify a transformation that will be applied after dequeuing the message.

Class

TDequeueOptions

Syntax

```
property Transformation: string;
```

Remarks

Use the Transformation property to specify a transformation that will be applied after dequeuing the message. Use it with Oracle 10 and higher.

See Also

- TEnqueueOptions.Transformation

5.16.1.1.2.9 Visibility Property

Used to specify the transactional behavior of the dequeue request.

Class

TDequeueOptions

Syntax

```
property Visibility: TQueueVisibility default qvOnCommit;
```

Remarks

Use Visibility property to specify the transactional behavior of the dequeue request.

See Also

- TEnqueueOptions.Visibility

Reserved.

5.16.1.1.2.10 WaitTimeout Property

Specifies the time to wait if there is no message matching the search criteria.

Class

TDequeueOptions

Syntax

```
property WaitTimeout: integer default AQ_FOREVER;
```

Remarks

Use WaitTimeout property to specify the wait time in seconds if there is currently no message matching the search criteria available. You can use constants AQ_FOREVER (the default value - wait forever) and AQ_NO_WAIT (do not wait).

Request Support DAC Forum Provide Feedback

5.16.1.2 TEnqueueOptions Class

A base class for setting default or special enqueue options for a queue.

For a list of all members of this type, see TEnqueueOptions members.

Unit

OraAQ

Syntax

```
TEnqueueOptions = class(TPersistent);
```

Remarks

Use the TEnqueueOptions class for setting default enqueue options for a queue or for using special enqueue options when calling TOraQueue.Enqueue or the TOraQueue.EnqueueArraymethod.

See Also

- TOraQueue.Enqueue
- TOraQueue.EnqueueArray
- TOraQueue.EnqueueOptions

 Request Support DAC Forum Provide Feedback

5.16.1.2.1 Members

TEnqueueOptions class overview.

Properties

Name	Description
DeliveryMode	Used to specify the type of the message that will be enqueued.
RelativeMessageId	Used to specify the message identifier of the message which is used in the sequence deviation operation.
SequenceDeviation	Used to specify whether the enqueued message should be dequeued before other messages that are in the queue already.
Transformation	Used to specify the transformation that will be applied before enqueuing a message.
Visibility	Used to specify the transactional behavior of the enqueue request.

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.16.1.2.2 Properties

Properties of the **TEnqueueOptions** class.

For a complete list of the **TEnqueueOptions** class members, see the TEnqueueOptions Members topic.

Public

Name	Description
RelativeMessageId	Used to specify the message identifier of the message which is used in the sequence deviation operation.

Published

Name	Description
DeliveryMode	Used to specify the type of the message that will be enqueued.
SequenceDeviation	Used to specify whether the enqueued message should be dequeued before other messages that are in the queue already.
Transformation	Used to specify the transformation that will be applied before enqueuing a message.
Visibility	Used to specify the transactional behavior of the enqueue request.

See Also

- TEnqueueOptions Class
- TEnqueueOptions Class Members

5.16.1.2.2.1 DeliveryMode Property

Used to specify the type of the message that will be enqueued.

Class

TEnqueueOptions

Syntax

```
property DeliveryMode: TQueueDeliveryMode default qdmPersistent;
```

Remarks

Use the DeliveryMode property to specify the type of the message that will be enqueued. Use it with Oracle 10 and higher.

See Also

- TDequeueOptions.DeliveryMode
- TQueueMessageProperties.DeliveryMode

Request Support DAC Forum Provide Feedback

5.16.1.2.2.2 RelativeMessageId Property

Used to specify the message identifier of the message which is used in the sequence deviation operation.

Class

TEnqueueOptions

Syntax

```
property RelativeMessageId: string;
```

Remarks

Use the RelativeMsgid property to specify the message identifier of the message which is used in the sequence deviation operation. Ignored if SequenceDeviation is not set to sdBefore.

See Also

- SequenceDeviation

Request Support DAC Forum Provide Feedback

5.16.1.2.2.3 SequenceDeviation Property

Used to specify whether the enqueued message should be dequeued before other messages that are in the queue already.

Class

TEnqueueOptions

Syntax

```
property SequenceDeviation: TQueueSequenceDeviation default
qsdNone;
```

See Also

- RelativeMessageId

Request Support DAC Forum Provide Feedback

5.16.1.2.2.4 Transformation Property

Used to specify the transformation that will be applied before enqueuing a message.

Class

TEnqueueOptions

Syntax

```
property Transformation: string;
```

Remarks

Use the Transformation property to specify a transformation that will be applied before enqueuing a message. Use it with Oracle 10 and higher.

See Also

- TDequeueOptions.Transformation

 Request Support DAC Forum Provide Feedback

5.16.1.2.2.5 Visibility Property

Used to specify the transactional behavior of the enqueue request.

Class

TEnqueueOptions

Syntax

```
property Visibility: TQueueVisibility default qvOnCommit;
```

Remarks

Use the Visibility property to specify the transactional behavior of the enqueue request.

See Also

- TDequeueOptions.Visibility

 Request Support DAC Forum Provide Feedback

5.16.1.3 TOraQueue Class

A component providing access to Oracle Streams Advanced Queuing.

For a list of all members of this type, see TOraQueue members.

Unit

OraAQ

Syntax

```
TOraQueue = class(TComponent);
```

Remarks

The TOraQueue component provides access to Oracle Streams Advanced Queuing. It allows to enqueue and dequeue messages and to listen to the queue.

Request Support DAC Forum Provide Feedback

5.16.1.3.1 Members

TOraQueue class overview.

Properties

Name	Description
AsyncNotification	Used to generate the OnMessage event each time when a new message is enqueued.
DequeueOptions	Used to set the default message dequeuing options.
EnqueueMessageProperties	Used to set the default enqueuing message properties.
EnqueueOptions	Used to set the default message enqueuing options.
PayloadArrayTypeName	Contains the type name of associative array, VARRAY, or nested table of queue payload type.
PayloadTypeName	Contains the payload type for the queue.
QueueName	Used to set the name of the Oracle queue to operation.
Session	Used to specify the session through which a queue will be controlled.

Methods

Name	Description
Dequeue	Overloaded. Dequeues messages.

DequeueArray	Dequeues an array of messages.
Enqueue	Overloaded. Enqueues messages.
EnqueueArray	Enqueues an array of messages.
Listen	Overloaded. Listens to one or more queues on behalf of the list of agents.

Events

Name	Description
OnMessage	Occurs when new messages are enqueued.

Request Support DAC Forum Provide Feedback

5.16.1.3.2 Properties

Properties of the **TOraQueue** class.

For a complete list of the **TOraQueue** class members, see the TOraQueue Members topic.

Public

Name	Description
PayloadArrayTypeName	Contains the type name of associative array, VARRAY, or nested table of queue payload type.
PayloadTypeName	Contains the payload type for the queue.

Published

Name	Description
AsyncNotification	Used to generate the OnMessage event each time when a new message is enqueued.

DequeueOptions	Used to set the default message dequeuing options.
EnqueueMessageProperties	Used to set the default enqueuing message properties.
EnqueueOptions	Used to set the default message enqueuing options.
QueueName	Used to set the name of the Oracle queue to operation.
Session	Used to specify the session through which a queue will be controlled.

See Also

- TOraQueue Class
- TOraQueue Class Members

Request Support DAC Forum Provide Feedback

5.16.1.3.2.1 AsyncNotification Property

Used to generate the OnMessage event each time when a new message is enqueued.

Class

TOraQueue

Syntax

```
property AsyncNotification: boolean default False;
```

Remarks

If the AsyncNotification propery is True, the TOraQueue component will generate the OnMessage event each time when a new message is enqueued.

When setting the AsyncNotification value to True, a message subscription is created in the queue. When setting the property value to False, the subscription is unregistered.

Active session is required to change the value of AsyncNotification, but TOraQueue doesn't need a session to get notifications. You can disconnect after setting the AsyncNotification value.

AsyncNotification property works only if OCI is initialized with the OCI_EVENTS flag. By default ODAC sets this flag if the OCI version is 9.0 or higher. If you need to use AsyncNotification with Oracle client 8.1, you should set the OCIEventsVersion variable from OraCall unit to 8100 in the initialization section of one of your program units.

AsyncNotification is not supported in Direct mode.

Note, the Oracle client version 8.1 has a bug when the NAMES.DEFAULT_DOMAIN parameter in the sqlnet.ora file is not working when the OCI_EVENTS flag is set.

See Also

- OnMessage

Request Support DAC Forum Provide Feedback

5.16.1.3.2.2 DequeueOptions Property

Used to set the default message dequeuing options.

Class

TOraQueue

Syntax

```
property DequeueOptions: TDequeueOptions;
```

Remarks

Use the DequeueOptions property to set the default message dequeuing options.

See Also

- TDequeueOptions
- EnqueueOptions

Request Support DAC Forum Provide Feedback

Reserved.

5.16.1.3.2.3 EnqueueMessageProperties Property

Used to set the default enqueuing message properties.

Class

TOraQueue

Syntax

```
property EnqueueMessageProperties: TQueueMessageProperties;
```

Remarks

Use the EnqueueMessageProperties property to set the default enqueuing message properties.

See Also

- TQueueMessageProperties

Request Support DAC Forum Provide Feedback

5.16.1.3.2.4 EnqueueOptions Property

Used to set the default message enqueuing options.

Class

TOraQueue

Syntax

```
property EnqueueOptions: TEnqueueOptions;
```

Remarks

Use the EnqueueOptions property to set the default message enqueuing options.

See Also

- TEnqueueOptions

- DequeueOptions

5.16.1.3.2.5 PayloadArrayTypeName Property

Contains the type name of associative array, VARRAY, or nested table of queue payload type.

Class

TOraQueue

Syntax

```
property PayloadArrayTypeName: string;
```

Remarks

The PayLoadArrayTypeName property contains the type name of associative array, VARRAY, or nested table of queue payload type. It is used by EnqueueArray and DequeueArray functions.

See Also

- EnqueueArray
- DequeueArray
- PayloadTypeName

5.16.1.3.2.6 PayloadTypeName Property

Contains the payload type for the queue.

Class

TOraQueue

Syntax

```
property PayloadTypeName: string;
```

Remarks

The PayloadTypeName property contains the payload type for the queue. PayloadType can be 'RAW' or the name of an object type. PayloadType value is used in Dequeue method. If value of the property is not set then Dequeue method will get payload type from the server before dequeuing message.

Request Support DAC Forum Provide Feedback

5.16.1.3.2.7 QueueName Property

Used to set the name of the Oracle queue to operation.

Class

TOraQueue

Syntax

```
property QueueName: string;
```

Remarks

Use the QueueName property to set the name of the Oracle queue to operation.

Request Support DAC Forum Provide Feedback

5.16.1.3.2.8 Session Property

Used to specify the session through which a queue will be controlled.

Class

TOraQueue

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session through which a queue will be controlled.

See Also

- TOraSession

Request Support DAC Forum Provide Feedback

5.16.1.3.3 Methods

Methods of the **TOraQueue** class.

For a complete list of the **TOraQueue** class members, see the TOraQueue Members topic.

Public

Name	Description
Dequeue	Overloaded. Dequeues messages.
DequeueArray	Dequeues an array of messages.
Enqueue	Overloaded. Enqueues messages.
EnqueueArray	Enqueues an array of messages.
Listen	Overloaded. Listens to one or more queues on behalf of the list of agents.

See Also

- TOraQueue Class
- TOraQueue Class Members

Request Support DAC Forum Provide Feedback

5.16.1.3.3.1 Dequeue Method

Dequeues messages.

Class

`TOraQueue`

Overload List

Name	Description
Dequeue(Payload: TOraObject; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeueus a message as a TOraObject.
Dequeue(Message: TQueueMessage; DequeueOptions: TDequeueOptions)	Dequeueus messages.
Dequeue(**out** Payload: TBytes; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeueus a message as array of Byte.
Dequeue(**out** Payload: **string**; MessageProperties: TQueueMessageProperties; DequeueOptions: TDequeueOptions)	Dequeues string messages.

Request Support DAC Forum Provide Feedback

Dequeueus a message as a TOraObject.

Class

TOraQueue

Syntax

```
function Dequeue(Payload: TOraObject; MessageProperties:
TQueueMessageProperties = nil; DequeueOptions: TDequeueOptions =
nil): TMessageId; overload;
```

Parameters

Payload

Holds a message content as a TOraObject.

MessageProperties

Holds the properties of a message that will be dequeued.

DequeueOptions

Holds the options for dequeueing messages.

Return Value

message ID as string.

Remarks

Use one of Dequeue method overloads to dequeue messages. Use overloads with string or TBytes payload for queues with the RAW payload type. For queues with the object payload type pass TOraObject instance with appropriate object type to Dequeue method. If DequeueOptions parameter was not specified, the TOraQueue.DequeueOptions property of TOraQueue component will be used. MessageProperties parameter will be filled with the properties of the dequeued message.

In Direct mode only queues with RAW payload are supported.

 Request Support DAC Forum Provide Feedback

Dequeueus messages.

Class

TOraQueue

Syntax

```
function Dequeue(Message: TQueueMessage; DequeueOptions:
TDequeueOptions = nil): TMessageId; overload;
```

Parameters

Message

Holds the message content.

DequeueOptions

Holds the options for dequeueing messages.

Return Value

message ID as string.

 Request Support DAC Forum Provide Feedback

Dequeueus a message as array of Byte.

Class

TOraQueue

Syntax

```
function Dequeue(out Payload: TBytes; MessageProperties:
TQueueMessageProperties = nil; DequeueOptions: TDequeueOptions =
nil): TMessageId; overload;
```

Parameters

Payload

Holds the message content as array of Byte.

MessageProperties

Holds the properties of a message that will be dequeued.

DequeueOptions

Holds the options for dequeueing messages.

Return Value

message ID as string.

Request Support DAC Forum Provide Feedback

Dequeues string messages.

Class

TOraQueue

Syntax

```
function Dequeue(out Payload: string; MessageProperties:
TQueueMessageProperties = nil; DequeueOptions: TDequeueOptions =
nil): TMessageId; overload;
```

Parameters

Payload

Holds a message as string.

MessageProperties

Holds the properties of a message that will be dequeued.

DequeueOptions

Holds the options for dequeueing messages.

Return Value

message ID as string.

See Also

- TOraQueue.OnMessage
- TOraQueue.DequeueOptions
- TOraQueue.Enqueue
- TOraQueue.DequeueArray
- TQueueMessage
- TDequeueOptions
- TQueueMessageProperties
- TOraObject

5.16.1.3.3.2 DequeueArray Method

Dequeues an array of messages.

Class

TOraQueue

Syntax

```
function DequeueArray(const MessageArray: array of TQueueMessage;
out DequeuedSize: integer; DequeueOptions: TDequeueOptions = nil):
TMessageIds;
```

Parameters

MessageArray
An array of messages to dequeue.

DequeuedSize
Returns the number of dequeued messages.

DequeueOptions
Holds the options for dequeueing messages.

Return Value
an array of message IDs.

Remarks

Use DequeueArray method to dequeue an array of messages. If DequeueOptions parameter was not specified, DequeueOptions property of TOraQueue component will be used. DequeuedSize returns the number of the dequeued messages.

See Also

- OnMessage
- TDequeueOptions
- Dequeue
- EnqueueArray
- TQueueMessage
- TQueueMessageProperties
- TOraObject

Request Support DAC Forum Provide Feedback

5.16.1.3.3.3 Enqueue Method

Enqueues messages.

Class

TOraQueue

Overload List

Name	Description
Enqueue(Payload: TOraObject; MessageProperties: TQueueMessageProperties; EnqueueOptions: TEnqueueOptions)	Enqueueus a message as TOraObject.
Enqueue(Message: TQueueMessage; EnqueueOptions: TEnqueueOptions)	Enqueues messages.
Enqueue(**const** Payload: TBytes; MessageProperties: TQueueMessageProperties;	Enqueueus a message as array of Byte.

EnqueueOptions: TEnqueueOptions)	
Enqueue(**const** Payload: **string**; MessageProperties: TQueueMessageProperties; EnqueueOptions: TEnqueueOptions)	Enqueues string messages.

Enqueueus a message as TOraObject.

Class

TOraQueue

Syntax

```
function Enqueue(Payload: TOraObject; MessageProperties:
TQueueMessageProperties = nil; EnqueueOptions: TEnqueueOptions =
nil): TMessageId; overload;
```

Parameters

Payload

a message content as a TOraObject.

MessageProperties

Holds the properties of the message which will be enqueued.

EnqueueOptions

Holds the options fore enqueueing messages.

Return Value

a message ID.

Remarks

Use one of the Enqueue method overloads to enqueue messages. Use overloads with string or TBytes payload for queues with the RAW payload type. For queues with object payload type pass the TOraObject instance with appropriate object type to the Enqueue method. If EnqueueOptions or MessageProperies parameters were not specified, the TOraQueue.EnqueueOptions and TOraQueue.EnqueueMessageProperties properties of TOraQueue component will be used.

In Direct mode only queues with RAW payload are supported.

Enqueues messages.

Class

TOraQueue

Syntax

```
function Enqueue(Message: TQueueMessage; EnqueueOptions: TEnqueueOptions = nil): TMessageId; overload;
```

Parameters

Message

Holds the message content.

EnqueueOptions

Holds the options fore enqueueing messages.

Return Value

a message ID.

Enqueueus a message as array of Byte.

Class

TOraQueue

Syntax

```
function Enqueue(const Payload: TBytes; MessageProperties: TQueueMessageProperties = nil; EnqueueOptions: TEnqueueOptions = nil): TMessageId; overload;
```

Parameters

Payload

Holds the message content as array of Byte.

MessageProperties

Holds the properties of the message which will be enqueued.

EnqueueOptions
Holds the options fore enqueueing messages.

Return Value
a message ID.

Enqueues string messages.

Class

TOraQueue

Syntax

```
function Enqueue(const Payload: string; MessageProperties: TQueueMessageProperties = nil; EnqueueOptions: TEnqueueOptions = nil): TMessageId; overload;
```

Parameters

Payload
Holds a message as a string.

MessageProperties
Holds the properties of the message which will be enqueued.

EnqueueOptions
Holds the options fore enqueueing messages.

Return Value
a message ID.

See Also

- TOraQueue.OnMessage
- TOraQueue.EnqueueOptions
- TOraQueue.EnqueueMessageProperties
- TOraQueue.Dequeue
- TQueueMessage
- TEnqueueOptions
- TQueueMessageProperties

- TOraObject

5.16.1.3.3.4 EnqueueArray Method

Enqueues an array of messages.

Class

TOraQueue

Syntax

```
function EnqueueArray(const MessageArray: array of TQueueMessage;
EnqueueOptions: TEnqueueOptions = nil): TMessageIds;
```

Parameters

MessageArray

Holds an array of messages to enqueue.

EnqueueOptions

Holds the options for enqueueing messages.

Return Value

an array of message IDs.

Remarks

Use the EnqueueArray method to enqueue an array of messages. If the EnqueueOptions parameter was not specified, the EnqueueOptions property of TOraQueue will be used. DequeuedSize returns the number of dequeued messages.

5.16.1.3.3.5 Listen Method

Listens to one or more queues on behalf of the list of agents.

Class

TOraQueue

Overload List

Name	Description
Listen(Agents: TQueueAgents; Agent: TQueueAgent; WaitTimeout: integer)	Listens to one or more queues on behalf of the list of agents.
Listen(Agents: TQueueAgents; ListDeliveryMode: TQueueDeliveryMode; Agent: TQueueAgent; **var** MessageDeliveryMode: TQueueDeliveryMode; WaitTimeout: integer)	Listens to one or more queues on behalf of the list of agents.

Request Support DAC Forum Provide Feedback

Listens to one or more queues on behalf of the list of agents.

Class

TOraQueue

Syntax

```
procedure Listen(Agents: TQueueAgents; Agent: TQueueAgent;
WaitTimeout: integer = AQ_FOREVER); overload;
```

Parameters

Agents
Holds the list of the agents.

Agent
Holds the agent name when monitoring multiconsumer queues.

WaitTimeout
Holds the amount of time to wait if there are no messages found.

Remarks

Call the Listen method to listen to one or more queues on behalf of the list of agents. You specify the queue to be monitored in the address field of each agent listed. You must also specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, only the first agent listed is returned. If there are no messages found when the wait time expires, an error is raised.

A successful return from the LISTEN call is only an indication that there is a message for one of the listed agents in one of the specified queues. The interested agent must still dequeue the relevant message.

Second overload with DeliveryMode is supported starting with Oracle 10. DeliveryMode possible values are described in TDequeueOptions.DeliveryMode topic.

See Also

- TDequeueOptions.DeliveryMode
- TQueueAgent
- TQueueAgents

Request Support DAC Forum Provide Feedback

Listens to one or more queues on behalf of the list of agents.

Class

TOraQueue

Syntax

```
procedure Listen(Agents: TQueueAgents; ListDeliveryMode:
TQueueDeliveryMode; Agent: TQueueAgent; var MessageDeliveryMode:
TQueueDeliveryMode; WaitTimeout: integer = AQ_FOREVER); overload;
```

Parameters

Agents
Holds the list of the agents.

ListDeliveryMode
Holds the message type (persistent, buffered messages or both).

Agent
Holds the agent name when monitoring multiconsumer queues.

MessageDeliveryMode
Holds the message type along with the queue and consumer for whom there is a message.

WaitTimeout

Holds the amount of time to wait if there are no messages found.

 Request Support DAC Forum Provide Feedback

5.16.1.3.4 Events

Events of the **TOraQueue** class.

For a complete list of the **TOraQueue** class members, see the TOraQueue Members topic.

Published

Name	Description
OnMessage	Occurs when new messages are enqueued.

See Also

- TOraQueue Class
- TOraQueue Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.3.4.1 OnMessage Event

Occurs when new messages are enqueued.

Class

TOraQueue

Syntax

```
property OnMessage: TQueueMessageEvent;
```

Remarks

Use OnMessage event handler to get notifications when new messages are enqueued. Set the AsyncNotification property to True to get OnMessage events.

To get the payload for the message assign MessageId value passed to the event handler to

the TDequeueOptions.MessageId property in DequeueOptions and then call TOraQueue.Dequeue.

See Also

- AsyncNotification
- TQueueMessageProperties
- DequeueOptions
- TDequeueOptions.MessageId

 Request Support DAC Forum Provide Feedback

5.16.1.4 TOraQueueAdmin Class

A component for managing queues in a database.

For a list of all members of this type, see TOraQueueAdmin members.

Unit

OraAQ

Syntax

```
TOraQueueAdmin = class(TComponent);
```

Remarks

Use the TOraQueueAdmin component to manage queues in a database.

Database user must have AQ_ADMINISTRATOR_ROLE to work with TOraQueueAdmin component.

 Request Support DAC Forum Provide Feedback

5.16.1.4.1 Members

TOraQueueAdmin class overview.

Properties

Name	Description
Comment	Used to get or set the user-specified description of the queue.
MaxRetries	Used to specify the number of attempts to dequeue message.
MultipleConsumers	Enables the possibility of using definite table for queues that can have multiple consumers for each message.
QueueName	Used to set the name of the Oracle queue to operate.
QueueTableName	Used to set or get the queue table for the queue.
QueueType	Indicates whether the queue being created is an exception queue or a normal queue.
RetentionTime	Used to set the time in seconds for which a message remains in the queue after being dequeued.
RetryDelay	Used to get or set delay time in seconds before the message that failed to be dequeued, will be scheduled to processing again.
Session	Used to specify the session through which queues will be managed.

Methods

Name	Description
AddSubscriber	Adds a subscriber to the queue.
AlterComment	Alters the user-defined queue description.
AlterMaxRetries	Alters the number of attempts to dequeue a message.

AlterPropagationSchedule	Alters parameters for a propagation schedule.
AlterQueue	Alters queue properties.
AlterRetentionTime	Alters the time in seconds during which a message remains in the queue after being dequeued.
AlterRetryDelay	Alters the delay time in seconds before the message, which failed to be dequeued, will be scheduled to processing again.
AlterSubscriber	Alters the rule and transformation properties of a queue subscriber.
CreateQueue	Creates a queue with the name, specified by the TOraQueueAdmin.QueueName property.
DisablePropagationSchedule	Disables a propagation schedule.
DropQueue	Drops the queue specified by the TOraQueueAdmin.QueueName property.
EnablePropagationSchedule	Enables a previously disabled propagation schedule.
GetSubscribers	Provides the list of queue subscribers.
GrantQueuePrivilege	Grants queue privilege to the grantee.
ReadQueueProperties	Reads the information about a queue specified by the TOraQueueAdmin.QueueName property from the database to a TOraQueueAdmin component.
RemoveSubscriber	Removes a subscriber from the queue.
RevokeQueuePrivilege	Revokes queue privilege from a grantee.

SchedulePropagation	Schedules the propagation of messages from the queue to a destination identified by a specific database link.
StartDequeue	Enables dequeueing on a queue.
StartEnqueue	Enables enqueueing on a queue.
StartQueue	Enables enqueueing, dequeueing, or both on a queue.
StopDequeue	Disables dequeueing on a queue.
StopEnqueue	Disables enqueueing on a queue.
StopQueue	Stops enqueueing, dequeueing, or both on a queue.
UnschedulePropagation	Unschedules previously scheduled propagation of messages from the queue to the specified destination.
VerifyQueueTypes	Verifies that the current queue and destination queue have the same type.

Request Support DAC Forum Provide Feedback

5.16.1.4.2 Properties

Properties of the **TOraQueueAdmin** class.

For a complete list of the **TOraQueueAdmin** class members, see the TOraQueueAdmin Members topic.

Published

Name	Description
Comment	Used to get or set the user-specified description of the queue.
MaxRetries	Used to specify the number

	of attempts to dequeue message.
MultipleConsumers	Enables the possibility of using definite table for queues that can have multiple consumers for each message.
QueueName	Used to set the name of the Oracle queue to operate.
QueueTableName	Used to set or get the queue table for the queue.
QueueType	Indicates whether the queue being created is an exception queue or a normal queue.
RetentionTime	Used to set the time in seconds for which a message remains in the queue after being dequeued.
RetryDelay	Used to get or set delay time in seconds before the message that failed to be dequeued, will be scheduled to processing again.
Session	Used to specify the session through which queues will be managed.

See Also

- TOraQueueAdmin Class
- TOraQueueAdmin Class Members

5.16.1.4.2.1 Comment Property

Used to get or set the user-specified description of the queue.

Class

TOraQueueAdmin

Syntax

```
property Comment: string;
```

Remarks

Call the Comment property to get or set the user-specified description of the queue.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.2 MaxRetries Property

Used to specify the number of attempts to dequeue message.

Class

TOraQueueAdmin

Syntax

```
property MaxRetries: integer default AQ_NOT_DEFINED;
```

Remarks

Use the MaxRetries property to specify the number of attempts to dequeue message.

See Also

- RetryDelay

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.3 MultipleConsumers Property

Enables the possibility of using definite table for queues that can have multiple consumers for each message.

Class

TOraQueueAdmin

Syntax

```
property MultipleConsumers: boolean default False;
```

Remarks

Should be set to True to use definite table for queues that can have multiple consumers for each message. False is the default value and means that queues, based on this table can have only one consumer for each message.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.4 QueueName Property

Used to set the name of the Oracle queue to operate.

Class

TOraQueueAdmin

Syntax

```
property QueueName: string;
```

Remarks

Use the QueueName property to set the name of the Oracle queue to operate.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.5 QueueTableName Property

Used to set or get the queue table for the queue.

Class

TOraQueueAdmin

Syntax

```
property QueueTableName: string;
```

Remarks

Use the QueueTableName property to set or get the queue table for the queue.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.6 QueueType Property

Indicates whether the queue being created is an exception queue or a normal queue.

Class

TOraQueueAdmin

Syntax

```
property QueueType: TQueueType default qtNormalQueue;
```

Remarks

Use the QueueType property to specify whether the queue being created is an exception queue or a normal queue. Only the dequeue operation is allowed in the exception queue. The default value is qtNormalQueue.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.7 RetentionTime Property

Used to set the time in seconds for which a message remains in the queue after being dequeued.

Class

TOraQueueAdmin

Syntax

```
property RetentionTime: integer default 0;
```

Remarks

Use the RetentionTime property to set the time in seconds for which a message remains in the queue after being dequeued. The default value is 0.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.8 RetryDelay Property

Used to get or set delay time in seconds before the message that failed to be dequeued, will be scheduled to processing again.

Class

TOraQueueAdmin

Syntax

```
property RetryDelay: integer default 0;
```

Remarks

Use the RetryDelay property to get or set delay time in seconds before the message that failed to be dequeued, will be scheduled to processing again. The default value is 0. Useless if MaxRetries property is set to 0.

 Request Support DAC Forum Provide Feedback

5.16.1.4.2.9 Session Property

Used to specify the session through which queues will be managed.

Class

TOraQueueAdmin

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session through which queues will be managed.

See Also

- TOraSession

5.16.1.4.3 Methods

Methods of the **TOraQueueAdmin** class.

For a complete list of the **TOraQueueAdmin** class members, see the TOraQueueAdmin Members topic.

Public

Name	Description
AddSubscriber	Adds a subscriber to the queue.
AlterComment	Alters the user-defined queue description.
AlterMaxRetries	Alters the number of attempts to dequeue a message.
AlterPropagationSchedule	Alters parameters for a propagation schedule.
AlterQueue	Alters queue properties.
AlterRetentionTime	Alters the time in seconds during which a message remains in the queue after being dequeued.
AlterRetryDelay	Alters the delay time in seconds before the message, which failed to be dequeued, will be scheduled to processing again.
AlterSubscriber	Alters the rule and transformation properties of a queue subscriber.
CreateQueue	Creates a queue with the name, specified by the TOraQueueAdmin.QueueName property.
DisablePropagationSchedule	Disables a propagation schedule.
DropQueue	Drops the queue specified by the

	TOraQueueAdmin.QueueName property.
EnablePropagationSchedule	Enables a previously disabled propagation schedule.
GetSubscribers	Provides the list of queue subscribers.
GrantQueuePrivilege	Grants queue privilege to the grantee.
ReadQueueProperties	Reads the information about a queue specified by the TOraQueueAdmin.QueueName property from the database to a TOraQueueAdmin component.
RemoveSubscriber	Removes a subscriber from the queue.
RevokeQueuePrivilege	Revokes queue privilege from a grantee.
SchedulePropagation	Schedules the propagation of messages from the queue to a destination identified by a specific database link.
StartDequeue	Enables dequeueing on a queue.
StartEnqueue	Enables enqueueing on a queue.
StartQueue	Enables enqueueing, dequeueing, or both on a queue.
StopDequeue	Disables dequeueing on a queue.
StopEnqueue	Disables enqueueing on a queue.
StopQueue	Stops enqueueing, dequeueing, or both on a queue.
UnschedulePropagation	Unschedules previously scheduled propagation of messages from the queue to the specified destination.
VerifyQueueTypes	Verifies that the current queue and destination

	queue have the same type.

See Also

- TOraQueueAdmin Class
- TOraQueueAdmin Class Members

Request Support DAC Forum Provide Feedback

5.16.1.4.3.1 AddSubscriber Method

Adds a subscriber to the queue.

Class

TOraQueueAdmin

Syntax

```
procedure AddSubscriber(Subscriber: TQueueAgent; const Rule:
string = ''; const Transformation: string = ''; QueueToQueue:
boolean = False; DeliveryMode: TQueueDeliveryMode = qdmPersistent);
```

Parameters

Subscriber

Holds an agent on whose behalf the subscription is being defined.

Rule

Holds a conditional expression based on the message properties, the message data properties, and PL/SQL functions.

Transformation

Holds the transformation that will be applied when this subscriber dequeues the message.

QueueToQueue

Is True, if propagation is from queue-to-queue.

DeliveryMode

Holds the delivery mode of the messages the subscriber is interested in. See TDequeueOptions.DeliveryMode for more information.

Remarks

Call the AddSubscriber method to add a subscriber to the queue.

See Also

- AlterSubscriber
- RemoveSubscriber
- TDequeueOptions.DeliveryMode
- TQueueAgent

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.2 AlterComment Method

Alters the user-defined queue description.

Class

TOraQueueAdmin

Syntax

```
procedure AlterComment(const Comment: string);
```

Parameters

Comment

Holds the user-defined queue description.

Remarks

Call the AlterComment method to alter the user-defined queue description.

See Also

- AlterQueue
- Comment

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.3 AlterMaxRetries Method

Alters the number of attempts to dequeue a message.

Class

TOraQueueAdmin

Syntax

```
procedure AlterMaxRetries(MaxRetries: integer);
```

Parameters

MaxRetries

Holds the number of attempts that can be taken to dequeue a message.

Remarks

Call the AlterMaxRetries method to alter the number of attempts to dequeue a message.

See Also

- AlterQueue
- MaxRetries

5.16.1.4.3.4 AlterPropagationSchedule Method

Alters parameters for a propagation schedule.

Class

TOraQueueAdmin

Syntax

```
procedure AlterPropagationSchedule(const Destination: string;
Duration: integer; const NextTime: string; Latency: integer;
const DestinationQueue: string = '');
```

Parameters

Destination

Holds the destination database link.

Duration

Holds the duration of the propagation window in seconds (NULL value means the propagation window is unscheduled forever or until the propagation).

NextTime

Holds the date function to compute the start of the next propagation window from the end of the current window.

Latency

Holds the maximum wait time in seconds in the propagation window for a message to be propagated after it is enqueued.

DestinationQueue

Holds the name of the destination queue. This parameter is supported starting with Oracle 10.

Remarks

Call the AlterPropagationSchedule method to alter parameters for a propagation schedule.

See Also

- SchedulePropagation
- DisablePropagationSchedule
- UnschedulePropagation

5.16.1.4.3.5 AlterQueue Method

Alters queue properties.

Class

TOraQueueAdmin

Syntax

```
procedure AlterQueue(MaxRetries: integer; RetryDelay: integer =
AQ_NOT_DEFINED; RetentionTime: integer = AQ_NOT_DEFINED; const
Comment: string = '');
```

Parameters

MaxRetries

Holds the number of attempts to dequeue message.

RetryDelay

Holds the delay time before the message that failed to be dequeued, will be scheduled to processing again.

RetentionTime

Holds the time for which a message remains in the queue after being dequeued.

Comment

Holds the user-specified description of the queue.

Remarks

Call the AlterQueue method to alter queue properties.

See Also

- MaxRetries
- RetryDelay
- RetentionTime
- Comment
- AlterMaxRetries
- AlterRetryDelay
- AlterRetentionTime
- AlterComment

Request Support DAC Forum Provide Feedback

5.16.1.4.3.6 AlterRetentionTime Method

Alters the time in seconds during which a message remains in the queue after being dequeued.

Class

TOraQueueAdmin

Syntax

```
procedure AlterRetentionTime(RetentionTime: integer);
```

Parameters

RetentionTime

Holds the time for which a message remains in the queue after being dequeued.

Remarks

Call the AlterRetentionTime method to alter the time in seconds during which a message remains in the queue after being dequeued.

See Also

- AlterQueue
- RetentionTime

Request Support DAC Forum Provide Feedback

5.16.1.4.3.7 AlterRetryDelay Method

Alters the delay time in seconds before the message, which failed to be dequeued, will be scheduled to processing again.

Class

TOraQueueAdmin

Syntax

```
procedure AlterRetryDelay(RetryDelay: integer);
```

Parameters

RetryDelay

Holds the delay time before the message that failed to be dequeued, will be scheduled to processing again.

Remarks

Call the AlterRetryDelay method to alter delay time in seconds before the message, which failed to be dequeued, will be scheduled to processing again.

See Also

- AlterQueue

- RetryDelay

5.16.1.4.3.8 AlterSubscriber Method

Alters the rule and transformation properties of a queue subscriber.

Class

TOraQueueAdmin

Syntax

```
procedure AlterSubscriber(Subscriber: TQueueAgent; const Rule:
string; const Transformation: string = '');
```

Parameters

Subscriber

Holds an agent on whose behalf the subscription is being defined.

Rule

Holds a conditional expression based on the message properties, the message data properties, and PL/SQL functions.

Transformation

Holds the transformation that will be applied when this subscriber dequeues the message.

Remarks

Call the AlterSubscriber method to alter the rule and transformation properties of a queue subscriber.

See Also

- AddSubscriber
- RemoveSubscriber
- TQueueAgent

5.16.1.4.3.9 CreateQueue Method

Creates a queue with the name, specified by the QueueName property.

Class

TOraQueueAdmin

Syntax

```
procedure CreateQueue(NonPersistent: boolean = False);
```

Parameters

NonPersistent

Holds True, if the created queue should be persistent. False otherwise.

Remarks

Call the CreateQueue method to create a queue with the name, specified by QueueName property. When creating a persistent queue, properties

- QueueTableName
- QueueType,
- MaxRetries,
- RetryDelay,
- RetentionTime,
- Comment

will be used as parameters for the new queue.

When creating a non-persistent queue, TOraQueueAdmin component uses properties MultipleConsumers and Comment.

5.16.1.4.3.10 DisablePropagationSchedule Method

Disables a propagation schedule.

Class

TOraQueueAdmin

Syntax

```
procedure DisablePropagationSchedule(const Destination: string;
const DestinationQueue: string = '');
```

Parameters

Destination

Holds the destination database link.

DestinationQueue

Holds the name of the destination queue.

Remarks

Call the DisablePropagationSchedule method to disable a propagation schedule. The DestinationQueue parameter is supported starting with Oracle 10.

See Also

- EnablePropagationSchedule
- SchedulePropagation
- UnschedulePropagation
- AlterPropagationSchedule

Request Support DAC Forum Provide Feedback

5.16.1.4.3.11 DropQueue Method

Drops the queue specified by the QueueName property.

Class

TOraQueueAdmin

Syntax

```
procedure DropQueue;
```

Remarks

Call the DropQueue method to drop the queue specified by the QueueName property.

See Also

- QueueName

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.12 EnablePropagationSchedule Method

Enables a previously disabled propagation schedule.

Class

TOraQueueAdmin

Syntax

```
procedure EnablePropagationSchedule(const Destination: string;
const DestinationQueue: string = '');
```

Parameters

Destination
Holds the destination database link.

DestinationQueue
Holds the name of the destination queue.

Remarks

Use the EnablePropagationSchedule method to enable a previously disabled propagation schedule. DestinationQueue parameter is supported starting with Oracle 10.

See Also

- DisablePropagationSchedule
- SchedulePropagation
- AlterPropagationSchedule

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.13 GetSubscribers Method

Provides the list of queue subscribers.

Class

TOraQueueAdmin

Syntax

```
procedure GetSubscribers(Subscribers: TQueueAgents);
```

Parameters

Subscribers

Holds a list of TOraAgent objects.

Remarks

Call the GetSubscribers method to get queue subscribers.

See Also

- AddSubscriber
- RemoveSubscriber
- TQueueAgents

5.16.1.4.3.14 GrantQueuePrivilege Method

Grants queue privilege to the grantee.

Class

TOraQueueAdmin

Syntax

```
procedure GrantQueuePrivilege(Privilege: TQueuePrivilege; const
Grantee: string; GrantOption: boolean = False);
```

Parameters

Privilege

Holds the kind of privilege.

Grantee

Holds the grantee name.

GrantOption

True, if the privilege will be granted with GRANT OPTION. False otherwise.

Remarks

Use the GrantQueuePrivilege method to grant queue privilege to the grantee. Privilege can be qpEnqueue (ENQUEUE privilege), qpDequeue (DEQUEUE privilege) or qpAll (both ENQUEUE and DEQUEUE privileges). If GrantOption parameter is set to True, the privilege will be granted with GRANT OPTION.

See Also

- RevokeQueuePrivilege

Request Support DAC Forum Provide Feedback

5.16.1.4.3.15 ReadQueueProperties Method

Reads the information about a queue specified by the QueueName property from the database to a TOraQueueAdmin component.

Class

TOraQueueAdmin

Syntax

```
procedure ReadQueueProperties;
```

Remarks

Call the ReadQueueProperties method to read the information about a queue specified by the QueueName property from the database to a TOraQueueAdmin component. After calling this method use properties

- QueueTableName
- QueueType,
- MaxRetries,

- RetryDelay,
- RetentionTime,
- Comment

to get queue parameters.

5.16.1.4.3.16 RemoveSubscriber Method

Removes a subscriber from the queue.

Class

TOraQueueAdmin

Syntax

```
procedure RemoveSubscriber(Subscriber: TQueueAgent);
```

Parameters

Subscriber

Holds an agent on whose behalf the subscription is being defined.

Remarks

Call the RemoveSubscriber method to remove a subscriber from the queue. All references to the subscriber in existing messages will be also removed.

See Also

- GetSubscribers
- AddSubscriber
- TQueueAgent

5.16.1.4.3.17 RevokeQueuePrivilege Method

Revokes queue privilege from a grantee.

Class

TOraQueueAdmin

Syntax

```
procedure RevokeQueuePrivilege(Privilege: TQueuePrivilege; const
Grantee: string);
```

Parameters

Privilege
Holds the kind of privilege.

Grantee
Holds the grantee name.

Remarks

Call the RevokeQueuePrivilege method to revoke queue privilege from a grantee. The privilege can be qpEnqueue (ENQUEUE privilege), qpDequeue (DEQUEUE privilege) or qpAll (both ENQUEUE and DEQUEUE privileges).

See Also

- GrantQueuePrivilege

5.16.1.4.3.18 SchedulePropagation Method

Schedules the propagation of messages from the queue to a destination identified by a specific database link.

Class

TOraQueueAdmin

Syntax

```
procedure SchedulePropagation(const Destination: string;
```

```
StartTime: TDateTime; Duration: integer; const NextTime: string;
Latency: integer; const DestinationQueue: string = '');
```

Parameters

Destination

Holds the destination database link.

StartTime

Holds the initial start time for the propagation window for messages from the queue to the destination.

Duration

Holds the duration of the propagation window in seconds.

NextTime

Holds the date function to compute the start of the next propagation window from the end of the current window.

Latency

Holds the maximum wait time in seconds in the propagation window for a message to be propagated after it is enqueued.

DestinationQueue

Holds the name of the destination queue. DestinationQueue parameter is supported starting with Oracle 10.

Remarks

Call the SchedulePropagation metod to schedule the propagation of messages from the queue to a destination identified by a specific database link.

See Also

- UnschedulePropagation
- AlterPropagationSchedule
- EnablePropagationSchedule

5.16.1.4.3.19 StartDequeue Method

Enables dequeueing on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StartDequeue;
```

Remarks

Call the StartDequeue method to enable dequeueing on a queue.

See Also

- StartQueue
- StartEnqueue
- StopDequeue

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.20 StartEnqueue Method

Enables enqueueing on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StartEnqueue;
```

Remarks

Call the StartEnqueue method to enable enqueueing on a queue.

See Also

- StartQueue
- StartDequeue
- StopEnqueue

 Request Support DAC Forum Provide Feedback

5.16.1.4.3.21 StartQueue Method

Enables enqueueing, dequeueing, or both on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StartQueue(Enqueue: boolean = True; Dequeue: boolean = True);
```

Parameters

Enqueue
True, if enqueueing is enabled. False otherwise.

Dequeue
True, if dequeueing is enabled. False otherwise.

Remarks

Call the StopQueue method to enable enqueueing, dequeueing, or both on a queue.

See Also

- StartEnqueue
- StartDequeue
- StopQueue

5.16.1.4.3.22 StopDequeue Method

Disables dequeueing on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StopDequeue(Wait: boolean = True);
```

Parameters

Wait

True, if waiting for the completion of the currently active queue transactions is enabled.

Remarks

Call the StopDequeue method to disable dequeueing on a queue. The Wait parameter specifies whether to wait for the completion of the currently active queue transactions.

See Also

- StopQueue
- StopEnqueue
- StartDequeue

Request Support DAC Forum Provide Feedback

5.16.1.4.3.23 StopEnqueue Method

Disables enqueueing on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StopEnqueue(Wait: boolean = True);
```

Parameters

Wait

True, if wait for the completion of the currently active queue transactions is enabled.

Remarks

Call the StopEnqueue method to disable enqueueing on a queue. The Wait parameter specifies whether to wait for the completion of the currently active queue transactions.

See Also

- StopQueue

- StopDequeue
- StartEnqueue

5.16.1.4.3.24 StopQueue Method

Stops enqueueing, dequeueing, or both on a queue.

Class

TOraQueueAdmin

Syntax

```
procedure StopQueue(Enqueue: boolean = True; Dequeue: boolean = True; Wait: boolean = True);
```

Parameters

Enqueue
True, if enqueueing is stopped. False otherwise.

Dequeue
True, if dequeueing is stopped. False otherwise.

Wait
True, if waiting for the completion of the currently active queue transactions is enabled.

Remarks

Call the StopQueue method to stop enqueueing, dequeueing, or both on a queue. The Wait parameter specifies whether to wait for the completion of the currently active queue transactions.

See Also

- StopEnqueue
- StopDequeue
- StartQueue

5.16.1.4.3.25 UnschedulePropagation Method

Unschedules previously scheduled propagation of messages from the queue to the specified destination.

Class

TOraQueueAdmin

Syntax

```
procedure UnschedulePropagation(const Destination: string; const
DestinationQueue: string = '');
```

Parameters

Destination

Holds the destination database link.

DestinationQueue

Holds the name of the destination queue. This parameter is supported starting with Oracle 10.

Remarks

Call the UnschedulePropagation method to unschedule previously scheduled propagation of messages from the queue to the specified destination (database link). DestinationQueue parameter is supported starting with Oracle 10.

See Also

- SchedulePropagation
- AlterPropagationSchedule
- DisablePropagationSchedule

5.16.1.4.3.26 VerifyQueueTypes Method

Verifies that the current queue and destination queue have the same type.

Class

TOraQueueAdmin

Syntax

```
function VerifyQueueTypes(const DestQueueName: string; const
Destination: string): boolean;
```

Parameters

DestQueueName

Holds the name of the destination queue.

Destination

Holds the destination database link.

Return Value

True, if the queues have the same type.

Remarks

Call the VerifyQueueTypes method to verify that the current queue and destination queue have the same type.

Request Support DAC Forum Provide Feedback

5.16.1.5 TOraQueueTable Class

A component managing queue tables in a database.

For a list of all members of this type, see TOraQueueTable members.

Unit

OraAQ

Syntax

```
TOraQueueTable = class(TComponent);
```

Remarks

Use the TOraQueueTable component to manage queue tables in a database.

Database user must have AQ_ADMINISTRATOR_ROLE to work with the TOraQueueTable component.

See Also

- TOraQueue, TOraQueueAdmin and TOraQueueTable Components
- TOraQueueAdmin
- TOraQueue

 Request Support DAC Forum Provide Feedback

5.16.1.5.1 Members

TOraQueueTable class overview.

Properties

Name	Description
Comment	Used to det or set a user-specified description of teh queue table.
Compatible	Used to define the lowest database version the queue is compatible with.
MessageGrouping	Used to specify the message grouping behavior in the queues based on this table.
MultipleConsumers	Used if a definite table for queues that can have multiple consumers for each message is needed.
PayloadTypeName	Used to get or set the object type name for the queue payload.
PrimaryInstance	Used to set the primary owner of the queue table.
QueueTableName	Used to get or set the queue table name.
SecondaryInstance	Used to set the secondary owner of the queue table.
Secure	Used for queue tables that will be used for secure queues.
Session	Used to specify the session through which a queue table will be controlled.

SortList	Used to specify the columns to be used as the sort key.
StorageClause	Used to get or set the table storage clause.

Methods

Name	Description
AlterComment	Alters queue table user-defined description.
AlterPrimaryInstance	Changes queue table primary_instance parameter.
AlterQueueTable	Changes existing queue table properties.
AlterSecondaryInstance	Changes the queue table secondary_instance parameter.
CreateQueueTable	Creates a queue table in a database.
DropQueueTable	Drops the queue table.
GrantSystemPrivilege	Grants system queue privilege.
MigrateQueueTable	Used to upgrade or downgrade a queue table to the desired Compatible parameter value.
PurgeQueueTable	Purges records from the queue table.
ReadQueueTableProperties	Reads information about a queue table specified by the QueueTableName property from the database to the TOraQueueTable component.
RevokeSystemPrivilege	Revokes system queue privilege.

5.16.1.5.2 Properties

Properties of the **TOraQueueTable** class.

For a complete list of the **TOraQueueTable** class members, see the TOraQueueTable Members topic.

Published

Name	Description
Comment	Used to det or set a user-specified description of teh queue table.
Compatible	Used to define the lowest database version the queue is compatible with.
MessageGrouping	Used to specify the message grouping behavior in the queues based on this table.
MultipleConsumers	Used if a definite table for queues that can have multiple consumers for each message is needed.
PayloadTypeName	Used to get or set the object type name for the queue payload.
PrimaryInstance	Used to set the primary owner of the queue table.
QueueTableName	Used to get or set the queue table name.
SecondaryInstance	Used to set the secondary owner of the queue table.
Secure	Used for queue tables that will be used for secure queues.
Session	Used to specify the session through which a queue table will be controlled.
SortList	Used to specify the columns to be used as the sort key.
StorageClause	Used to get or set the table storage clause.

See Also

- TOraQueueTable Class
- TOraQueueTable Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.1 Comment Property

Used to det or set a user-specified description of teh queue table.

Class

TOraQueueTable

Syntax

```
property Comment: string;
```

Remarks

Use The Comment property to get user-specified description of the queue table or to set it. It is used by the CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.2 Compatible Property

Used to define the lowest database version the queue is compatible with.

Class

TOraQueueTable

Syntax

```
property Compatible: TQueueCompatible default qcDefault;
```

Remarks

Use the Compatible property to determine the lowest database version with which the queue is compatible. The default value is qcDefault. The meaning of this value depends on the database compatible mode. For more information read the Oracle documentation. This property is used by the CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.3 MessageGrouping Property

Used to specify the message grouping behavior in the queues based on this table.

Class

TOraQueueTable

Syntax

```
property MessageGrouping: TQueueMessageGrouping default qmgNone;
```

Remarks

Use the MessageGrouping property to determine message grouping behavior in the queues based on this table. mgNone means that each queue message is treated individually, mgTransactional means that messages, enqueued in one transaction, will belong to the same group. The default value is mgNone. This property is used by the CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.4 MultipleConsumers Property

Used if a definite table for queues that can have multiple consumers for each message is needed.

Class

TOraQueueTable

Syntax

```
property MultipleConsumers: boolean default False;
```

Remarks

The MultipleConsumers property should be set to True to use a definite table for queues that can have multiple consumers for each message. False is the default value and means that queues, based on this table can have only one consumer for each message. This property is used by the CreateQueueTable method.

See Also

- CreateQueueTable

Request Support DAC Forum Provide Feedback

5.16.1.5.2.5 PayloadTypeName Property

Used to get or set the object type name for the queue payload.

Class

TOraQueueTable

Syntax

```
property PayloadTypeName: string stored IsPayloadTypeNameStored;
```

Remarks

Use the PayloadTypeName property to get or set the object type name for the queue payload. This property is used by the CreateQueueTable method. This property is set to 'RAW' for RAW payload.

See Also

- CreateQueueTable

5.16.1.5.2.6 PrimaryInstance Property

Used to set the primary owner of the queue table.

Class

TOraQueueTable

Syntax

```
property PrimaryInstance: integer default 0;
```

Remarks

Use the PrimaryInstance property to get or set primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value for primary instance is 0, what means that queue monitor scheduling and propagation will be done in any available instance. This property is used by CreateQueueTable method.

See Also

- CreateQueueTable

5.16.1.5.2.7 QueueTableName Property

Used to get or set the queue table name.

Class

TOraQueueTable

Syntax

```
property QueueTableName: string;
```

Remarks

Use the QueueTableName property to get or set the queue table name. This property is used by the CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.8 SecondaryInstance Property

Used to set the secondary owner of the queue table.

Class

TOraQueueTable

Syntax

```
property SecondaryInstance: integer default 0;
```

Remarks

The queue table fails over the secondary instance if the primary instance is not available. The default value is 0. It means that the queue table will fail over any available instance. This property is used by CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.9 Secure Property

Used for queue tables that will be used for secure queues.

Class

TOraQueueTable

Syntax

```
property Secure: boolean default False;
```

Remarks

The Secure property must be True for queue tables that will be used for secure queues. It is used by the CreateQueueTable method.

See Also

- CreateQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.10 Session Property

Used to specify the session through which a queue table will be controlled.

Class

TOraQueueTable

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session through which a queue table will be controlled.

See Also

- TOraSession

 Request Support DAC Forum Provide Feedback

5.16.1.5.2.11 SortList Property

Used to specify the columns to be used as the sort key.

Class

TOraQueueTable

Syntax

```
property SortList: TQueueSortList default qslDefault;
```

Remarks

Use the SortList property to specify the columns to be used as the sort key in the ascending order.

See Also

- CreateQueueTable

5.16.1.5.2.12 StorageClause Property

Used to get or set the table storage clause.

Class

TOraQueueTable

Syntax

```
property StorageClause: string;
```

Remarks

Use the StorageClause property to get or set the table storage clause. It is used by the CreateQueueTable method. May contain any SQL code that can be in the CREATE TABLE storage clause.

See Also

- CreateQueueTable

5.16.1.5.3 Methods

Methods of the **TOraQueueTable** class.

For a complete list of the **TOraQueueTable** class members, see the TOraQueueTable Members topic.

Public

Name	Description
AlterComment	Alters queue table user-defined description.
AlterPrimaryInstance	Changes queue table primary_instance parameter.
AlterQueueTable	Changes existing queue table properties.
AlterSecondaryInstance	Changes the queue table secondary_instance parameter.
CreateQueueTable	Creates a queue table in a database.
DropQueueTable	Drops the queue table.
GrantSystemPrivilege	Grants system queue privilege.
MigrateQueueTable	Used to upgrade or downgrade a queue table to the desired Compatible parameter value.
PurgeQueueTable	Purges records from the queue table.
ReadQueueTableProperties	Reads information about a queue table specified by the QueueTableName property from the database to the TOraQueueTable component.
RevokeSystemPrivilege	Revokes system queue privilege.

See Also

- TOraQueueTable Class
- TOraQueueTable Class Members

5.16.1.5.3.1 AlterComment Method

Alters queue table user-defined description.

Class

TOraQueueTable

Syntax

```
procedure AlterComment(const Comment: string);
```

Parameters

Comment

Holds the user-defined description of the queue-table.

Remarks

Call the AlterComment method to alter queue table user-defined description.

See Also

- Comment
- AlterQueueTable

5.16.1.5.3.2 AlterPrimaryInstance Method

Changes queue table primary_instance parameter.

Class

TOraQueueTable

Syntax

```
procedure AlterPrimaryInstance(PrimaryInstance: integer);
```

Parameters

PrimaryInstance
Holds the primary owner of the queue table.

Remarks

Call the AlterPrimaryInstance method to alter queue table primary_instance parameter.

See Also

- PrimaryInstance
- AlterQueueTable

Request Support DAC Forum Provide Feedback

5.16.1.5.3.3 AlterQueueTable Method

Changes existing queue table properties.

Class

TOraQueueTable

Syntax

```
procedure AlterQueueTable(const Comment: string; PrimaryInstance:
integer = AQ_NOT_DEFINED; SecondaryInstance: integer =
AQ_NOT_DEFINED);
```

Parameters

Comment
Holds the user-defined description of the queue-table.

PrimaryInstance
Holds the primary owner of the queue table.

SecondaryInstance
Holds the secondary owner of the queue table.

Remarks

Call the AlterQueueTable method to alter existing queue table properties. You can alter the following properties: Comment, PrimaryInstance, SecondaryInstance.

See Also

- Comment
- PrimaryInstance
- SecondaryInstance
- AlterComment
- AlterSecondaryInstance
- AlterPrimaryInstance

 Request Support DAC Forum Provide Feedback

5.16.1.5.3.4 AlterSecondaryInstance Method

Changes the queue table secondary_instance parameter.

Class

TOraQueueTable

Syntax

```
procedure AlterSecondaryInstance(SecondaryInstance: integer);
```

Parameters

SecondaryInstance
Holds the secondary owner of the queue table.

Remarks

Call the AlterSecondaryInstance method to alter queue table secondary_instance parameter.

See Also

- SecondaryInstance
- AlterQueueTable

 Request Support DAC Forum Provide Feedback

5.16.1.5.3.5 CreateQueueTable Method

Creates a queue table in a database.

Class

TOraQueueTable

Syntax

```
procedure CreateQueueTable;
```

Remarks

Call the CreateQueueTable method to create a queue table in a database. The QueueTableName property of the TOraQueueTable component will be used as a name for the queue table. Properties

- StorageClause,
- PayloadTypeName,
- SortList,
- MultipleConsumers,
- MessageGrouping,
- Comment,
- PrimaryInstance,
- SecondaryInstance,
- Compatible,
- Secure

will be used as parameters for a new queue table.

5.16.1.5.3.6 DropQueueTable Method

Drops the queue table.

Class

TOraQueueTable

Syntax

```
procedure DropQueueTable(Force: boolean = False);
```

Parameters

Force

If True, stops and dropps all queues in the table automatically. Otherwise the table will not be dropped if there are any queues in it.

Remarks

Call the DropQueueTable method to drop the queue table. If the Force parameter is set to False, the table will not be dropped if there are any queues in the table. Otherwise all queues will be stopped and dropped automatically.

Request Support DAC Forum Provide Feedback

5.16.1.5.3.7 GrantSystemPrivilege Method

Grants system queue privilege.

Class

TOraQueueTable

Syntax

```
procedure GrantSystemPrivilege(Privilege: TQueueSystemPrivilege;
const Grantee: string; AdminOption: boolean = False);
```

Parameters

Privilege

Holds a system privilege to a grant.

Grantee

Holds the user name, a role, or a PUBLIC role.

AdminOption
Specifies whether the privilege will be granted with ADMIN OPTION.

Remarks

Call the GrantSystemPrivilege method to grant system queue privilege. Privilege parameter is a system privilege to grant. Can be qspEnqueueAny (ENQUEUE_ANY privilege), qspDequeueAny (DEQUEUE_ANY privilege), qspManageAny (MANAGE_ANY privilege). Grantee parameter is a grantee - can be a user, a role, or the PUBLIC role. AdminOption parameter specifies whether the privilege will be granted with ADMIN OPTION.

See Also

- RevokeSystemPrivilege

Request Support DAC Forum Provide Feedback

5.16.1.5.3.8 MigrateQueueTable Method

Used to upgrade or downgrade a queue table to the desired Compatible parameter value.

Class

TOraQueueTable

Syntax

```
procedure MigrateQueueTable(Compatible: TQueueCompatible);
```

Parameters

Compatible
Holds the lowest database version the queue is compatible with.

Remarks

Call the MigrateQueueTable method to upgrade or downgrade a queue table to the desired Compatible parameter value.

See Also

- Compatible

Request Support DAC Forum Provide Feedback

Reserved.

5.16.1.5.3.9 PurgeQueueTable Method

Purges records from the queue table.

Class

TOraQueueTable

Syntax

```
procedure PurgeQueueTable(const PurgeCondition: string; Block:
boolean = False; DeliveryMode: TQueueDeliveryMode = qdmPersistent);
```

Parameters

PurgeCondition

Specifies the purge condition, which must be in the format of a SQL WHERE clause and is based on the columns of aq$queue_table_name view.

Block

Specifies whether to hold an exclusive lock on all the queues in the queue table while purging the queue table.

DeliveryMode

Specifies which type of messages should be purged. Possible values are described in TDequeueOptions.DeliveryMode topic

Remarks

Call the PurgeQueueTable method to purge records from the queue table.

See Also

- TDequeueOptions.DeliveryMode

Request Support DAC Forum Provide Feedback

5.16.1.5.3.10 ReadQueueTableProperties Method

Reads information about a queue table specified by the QueueTableName property from the database to the TOraQueueTable component.

Class

TOraQueueTable

Syntax

```
procedure ReadQueueTableProperties;
```

Remarks

Call the ReadQueueTableProperties method to read information about a queue table specified by the QueueTableName property from the database to the TOraQueueTable component. After calling this method use properties

- PayloadTypeName,
- SortList,
- MultipleConsumers,
- MessageGrouping,
- Comment,
- PrimaryInstance,
- SecondaryInstance,
- Compatible,
- Secure

to get queue table parameters.

Request Support DAC Forum Provide Feedback

5.16.1.5.3.11 RevokeSystemPrivilege Method

Revokes system queue privilege.

Class

TOraQueueTable

Syntax

```
procedure RevokeSystemPrivilege(Privilege: TQueueSystemPrivilege;
const Grantee: string);
```

Parameters

Privilege
Holds the system privilege to revoke.

Grantee
Holds the user name, a role, or a PUBLIC role.

Remarks

Call the RevokeSystemPrivilege method to revoke system queue privilege. Privilege parameter is a system privilege to revoke. Can be qspEnqueueAny (ENQUEUE_ANY privilege), qspDequeueAny (DEQUEUE_ANY privilege), qspManageAny (MANAGE_ANY privilege). Grantee parameter is a grantee - can be a user, a role, or the PUBLIC role.

Request Support DAC Forum Provide Feedback

5.16.1.6 TQueueAgent Class

A class reperesenting a producer or a consumer of a queue message.

For a list of all members of this type, see TQueueAgent members.

Unit

OraAQ

Syntax

```
TQueueAgent = class(TCollectionItem);
```

Remarks

The TQueueAgent class represents a producer or a consumer of a queue message. Use the TQueueAgent class to set producers or consumers of queue messages, queue subscribers etc.

See Also

- TQueueAgents
- TQueueMessageProperties.SenderId
- TOraQueue.Listen
- TOraQueueAdmin.AddSubscriber

- TOraQueueAdmin.RemoveSubscriber
- TOraQueueAdmin.AlterSubscriber

 Request Support DAC Forum Provide Feedback

5.16.1.6.1 Members

TQueueAgent class overview.

Properties

Name	Description
Address	Used to get or set the recipient protocol-specific address.
Name	Used to get or set a name of the queue agent.
Protocol	Used to set a protocol to interpret the address and propagate the message.

 Request Support DAC Forum Provide Feedback

5.16.1.6.2 Properties

Properties of the **TQueueAgent** class.

For a complete list of the **TQueueAgent** class members, see the TQueueAgent Members topic.

Published

Name	Description
Address	Used to get or set the recipient protocol-specific address.
Name	Used to get or set a name of the queue agent.
Protocol	Used to set a protocol to interpret the address and

	propagate the message.

See Also

- TQueueAgent Class
- TQueueAgent Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.6.2.1 Address Property

Used to get or set the recipient protocol-specific address.

Class

TQueueAgent

Syntax

```
property Address: string;
```

Remarks

Use the Address property to get or set the recipient protocol-specific address. If the Protocol property is 0, then the address should have the [schema.]queue[@dblink] form.

 Request Support DAC Forum Provide Feedback

5.16.1.6.2.2 Name Property

Used to get or set a name of the queue agent.

Class

TQueueAgent

Syntax

```
property Name: string;
```

Remarks

Use the Name property to get or set a name of the queue agent.

 Request Support DAC Forum Provide Feedback

5.16.1.6.2.3 Protocol Property

Used to set a protocol to interpret the address and propagate the message.

Class

TQueueAgent

Syntax

```
property Protocol: integer default 0;
```

Remarks

Use the Protocol property to set a protocol to interpret the address and propagate the message.

 Request Support DAC Forum Provide Feedback

5.16.1.7 TQueueAgents Class

A class holding a collection of the TQueueAgent objects.

For a list of all members of this type, see TQueueAgents members.

Unit

OraAQ

Syntax

```
TQueueAgents = class(TOwnedCollection);
```

Remarks

Each TQueueAgents holds a collection of the TQueueAgent objects. TQueueAgents maintains an index of the columns in its Items array. The Count property contains the number of columns in the collection.

See Also

- TQueueAgent
- TQueueMessageProperties.RecipientList
- TOraQueue.Listen
- TOraQueueAdmin.GetSubscribers

 Request Support DAC Forum Provide Feedback

5.16.1.7.1 Members

TQueueAgents class overview.

Properties

Name	Description
Items	Used to access individual columns.

Methods

Name	Description
Add	Adds a TQueueAgent object to the collection.
Insert	Inserts a TQueueAgent object in the TQueueAgents collection to the required place.

 Request Support DAC Forum Provide Feedback

5.16.1.7.2 Properties

Properties of the **TQueueAgents** class.

For a complete list of the **TQueueAgents** class members, see the TQueueAgents Members topic.

Public

Name	Description
Items	Used to access individual columns.

See Also

- TQueueAgents Class
- TQueueAgents Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.7.2.1 Items Property(Indexer)

Used to access individual columns.

Class

TQueueAgents

Syntax

```
property Items[Index: integer]: TQueueAgent; default;
```

Parameters

Index

Holds the index of a TQueueAgent object.

Remarks

Use the Items property to access individual columns. The value of the Index parameter corresponds to the Index property of TQueueAgent.

See Also

- TQueueAgent

 Request Support DAC Forum Provide Feedback

5.16.1.7.3 Methods

Methods of the **TQueueAgents** class.

For a complete list of the **TQueueAgents** class members, see the TQueueAgents Members topic.

Public

Name	Description
Add	Adds a TQueueAgent object to the collection.
Insert	Inserts a TQueueAgent object in the TQueueAgents collection to the required place.

See Also

- TQueueAgents Class
- TQueueAgents Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.7.3.1 Add Method

Adds a TQueueAgent object to the collection.

Class

TQueueAgents

Syntax

```
function Add: TQueueAgent;
```

Return Value

a TQueueAgent object with empty properties.

Remarks

Call the Add method to add a TQueueAgent object with empty properties to the collection.

 Request Support DAC Forum Provide Feedback

5.16.1.7.3.2 Insert Method

Inserts a TQueueAgent object in the TQueueAgents collection to the required place.

Class

TQueueAgents

Syntax

```
function Insert(Index: Integer): TQueueAgent;
```

Parameters

Index

Holds the index of the place to insert a TQueueAgent object to.

Return Value

a TQueueAgent object.

Remarks

Call the Insert method to insert a TQueueAgent object in the TQueueAgents collection to the required place, specified with Index parameter.

Request Support DAC Forum Provide Feedback

5.16.1.8 TQueueMessage Class

A class representing a queue message.

For a list of all members of this type, see TQueueMessage members.

Unit

OraAQ

Syntax

```
TQueueMessage = class(System.TObject);
```

Remarks

The TQueueMessage class represents a queue message. Use the TQueueMessage class for setting or reading message parameters and payload when sending or receiving a queue message.

See Also

- TOraQueue.Dequeue
- TOraQueue.DequeueArray
- TOraQueue.Enqueue
- TOraQueue.EnqueueArray

 Request Support DAC Forum Provide Feedback

5.16.1.8.1 Members

TQueueMessage class overview.

Properties

Name	Description
MessageId	Used to provide a message ID.
MessageProperties	Used to get or set queue the queue message parameters.
RawPayload	Used to set or get payload of the queue message as RAW payload.
StringPayload	Used to set or get the payload of a queue message as string payload.

 Request Support DAC Forum Provide Feedback

5.16.1.8.2 Properties

Properties of the **TQueueMessage** class.

For a complete list of the **TQueueMessage** class members, see the TQueueMessage

Members topic.

Public

Name	Description
MessageId	Used to provide a message ID.
MessageProperties	Used to get or set queue the queue message parameters.
RawPayload	Used to set or get payload of the queue message as RAW payload.
StringPayload	Used to set or get the payload of a queue message as string payload.

See Also

- TQueueMessage Class
- TQueueMessage Class Members

 Request Support DAC Forum Provide Feedback

5.16.1.8.2.1 MessageId Property

Used to provide a message ID.

Class

TQueueMessage

Syntax

```
property MessageId: TMessageId;
```

Remarks

Use the MessageId property to get the message ID for a message.

 Request Support DAC Forum Provide Feedback

5.16.1.8.2.2 MessageProperties Property

Used to get or set queue the queue message parameters.

Class

TQueueMessage

Syntax

```
property MessageProperties: TQueueMessageProperties;
```

Remarks

Use the MessageProperties property to get or set queue the queue message parameters that AQ uses to manage individual messages.

See Also

- TQueueMessageProperties

5.16.1.8.2.3 Raw Payload Property

Used to set or get payload of the queue message as RAW payload.

Class

TQueueMessage

Syntax

```
property RawPayload: TBytes;
```

Remarks

Use the RawPayload method to set or get payload of the queue message as RAW payload.

5.16.1.8.2.4 StringPayload Property

Used to set or get the payload of a queue message as string payload.

Class

TQueueMessage

Syntax

```
property StringPayload: string;
```

Remarks

Use the StringPayload property to set or get the payload of a queue message as string payload.

Request Support DAC Forum Provide Feedback

5.16.1.9 TQueueMessageProperties Class

A class for setting or providing queue message properties.

For a list of all members of this type, see TQueueMessageProperties members.

Unit

OraAQ

Syntax

```
TQueueMessageProperties = class(TPersistent);
```

Remarks

Use the TQueueMessageProperties class to set or get queue message properties.

See Also

- TQueueMessage.MessageProperties
- TOraQueue.EnqueueMessageProperties
- TOraQueue.OnMessage

- TOraQueue.Dequeue
- TOraQueue.DequeueArray
- TOraQueue.Enqueue
- TOraQueue.EnqueueArray

 Request Support DAC Forum Provide Feedback

5.16.1.9.1 Members

TQueueMessageProperties class overview.

Properties

Name	Description
Attempts	Contains the number of attempts that have been made to dequeue a message.
Correlation	Contains the identifier supplied by the message producer at enqueue time.
Delay	Contains the delay of the enqueued message.
DeliveryMode	Used to indicate whether the message will be enqueued as buffered or persistent and to specify what kinds of messages should be dequeued.
EnqueueTime	Contains the time when a message was enqueued.
ExceptionQueue	Contains the name of the queue into which the message is moved if it cannot be processed successfully.
Expiration	Contains the expiration of a message.
OriginalMessageId	Contains the parameter for propagating messages.

Priority	Contains the message priority.
RecipientList	Contains the list of agents that receive a message.
SenderId	Contains application-sender identification.
State	Contains the state of a message.
TransactionGroup	Contains the transaction_group for the dequeued message.

Request Support DAC Forum Provide Feedback

5.16.1.9.2 Properties

Properties of the **TQueueMessageProperties** class.

For a complete list of the **TQueueMessageProperties** class members, see the TQueueMessageProperties Members topic.

Public

Name	Description
Attempts	Contains the number of attempts that have been made to dequeue a message.
Correlation	Contains the identifier supplied by the message producer at enqueue time.
EnqueueTime	Contains the time when a message was enqueued.
OriginalMessageId	Contains the parameter for propagating messages.
State	Contains the state of a message.
TransactionGroup	Contains the transaction_group for the dequeued message.

Published

Name	Description
Delay	Contains the delay of the enqueued message.
DeliveryMode	Used to indicate whether the message will be enqueued as buffered or persistent and to specify what kinds of messages should be dequeued.
ExceptionQueue	Contains the name of the queue into which the message is moved if it cannot be processed successfully.
Expiration	Contains the expiration of a message.
Priority	Contains the message priority.
RecipientList	Contains the list of agents that receive a message.
SenderId	Contains application-sender identification.

See Also

- TQueueMessageProperties Class
- TQueueMessageProperties Class Members

Request Support DAC Forum Provide Feedback

5.16.1.9.2.1 Attempts Property

Contains the number of attempts that have been made to dequeue a message.

Class

TQueueMessageProperties

Syntax

```
property Attempts: integer;
```

Remarks

The Attempts property contains the number of attempts that have been made to dequeue a message.

Request Support DAC Forum Provide Feedback

5.16.1.9.2.2 Correlation Property

Contains the identifier supplied by the message producer at enqueue time.

Class

TQueueMessageProperties

Syntax

```
property Correlation: string;
```

Remarks

The Correlation property contains the identifier supplied by the message producer at enqueue time.

Request Support DAC Forum Provide Feedback

5.16.1.9.2.3 Delay Property

Contains the delay of the enqueued message.

Class

TQueueMessageProperties

Syntax

```
property Delay: integer default AQ_NO_DELAY;
```

Remarks

The Delay property contains the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. The default value is

AQ_NO_DELAY. That means the message will be available for immediate dequeuing.

Request Support DAC Forum Provide Feedback

5.16.1.9.2.4 DeliveryMode Property

Used to indicate whether the message will be enqueued as buffered or persistent and to specify what kinds of messages should be dequeued.

Class

TQueueMessageProperties

Syntax

```
property DeliveryMode: TQueueDeliveryMode default qdmPersistent;
```

Remarks

Use the DeliveryMode property to indicate whether the message will be enqueued as buffered or persistent when enqueueing it, and to specify what kinds of messages should be dequeued (buffered, persistent or both), when dequeueing messages. qdmPersistentOrBuffered value can be set only when dequeuing a message.

See Also

- TEnqueueOptions.DeliveryMode
- TDequeueOptions.DeliveryMode

Request Support DAC Forum Provide Feedback

5.16.1.9.2.5 EnqueueTime Property

Contains the time when a message was enqueued.

Class

TQueueMessageProperties

Syntax

```
property EnqueueTime: TDateTime;
```

Remarks

The EnqueueTime property contains the time when a message was enqueued.

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.6 ExceptionQueue Property

Contains the name of the queue into which the message is moved if it cannot be processed successfully.

Class

TQueueMessageProperties

Syntax

```
property ExceptionQueue: string;
```

Remarks

The ExceptionQueue property contains the name of the queue into which the message is moved if it cannot be processed successfully.

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.7 Expiration Property

Contains the expiration of a message.

Class

TQueueMessageProperties

Syntax

```
property Expiration: integer default AQ_NEVER;
```

Remarks

The Expiration property contains the expiration of a message in seconds. It is the duration the message is available for dequeuing. The default value is AQ_NEVER. It means that the message will never expire.

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.8 OriginalMessageId Property

Contains the parameter for propagating messages.

Class

TQueueMessageProperties

Syntax

```
property OriginalMessageId: string;
```

Remarks

The OriginalMessageId property contains the parameter used by Oracle Streams AQ for propagating messages.

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.9 Priority Property

Contains the message priority.

Class

TQueueMessageProperties

Syntax

```
property Priority: integer default 1;
```

Remarks

The Priority property contains the message priority. A smaller number indicates higher priority. The priority can be any number, including negative numbers.

5.16.1.9.2.10 RecipientList Property

Contains the list of agents that receive a message.

Class

TQueueMessageProperties

Syntax

```
property RecipientList: TQueueAgents stored
IsRecipientListStored;
```

Remarks

The RecipientList property contains the list of agents that receive a message.

See Also

- TQueueAgents

5.16.1.9.2.11 SenderId Property

Contains application-sender identification.

Class

TQueueMessageProperties

Syntax

```
property SenderId: TQueueAgent;
```

Remarks

The SenderId property contains application-sender identification specified at the enqueue time by the message producer.

See Also

- TQueueAgent

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.12 State Property

Contains the state of a message.

Class

TQueueMessageProperties

Syntax

```
property State: TQueueMessageState;
```

Remarks

The State property contains the state of a message.

 Request Support DAC Forum Provide Feedback

5.16.1.9.2.13 TransactionGroup Property

Contains the transaction_group for the dequeued message.

Class

TQueueMessageProperties

Syntax

```
property TransactionGroup: string;
```

Remarks

The TransactionGroup property contains the transaction_group for the dequeued message.

Messages belonging to the same transaction group will have the same value for this attribute.

 Request Support DAC Forum Provide Feedback

Reserved.

5.16.2 Types

Types in the **OraAQ** unit.

Types

Name	Description
TQueueMessageEvent	This type is used for the TOraQueue.OnMessage event.

Request Support DAC Forum Provide Feedback

5.16.2.1 TQueueMessageEvent Procedure Reference

This type is used for the TOraQueue.OnMessage event.

Unit

OraAQ

Syntax

```
TQueueMessageEvent = procedure (Sender: TOraQueue; const
MessageId: string; const MessageProperties:
TQueueMessageProperties) of object;
```

Parameters

Sender
An object that raised the event.

MessageId
Holds the payload for the message.

MessageProperties
Holds the message properties.

Request Support DAC Forum Provide Feedback

5.16.3 Enumerations

Enumerations in the **OraAQ** unit.

Enumerations

Name	Description
TDequeueMode	Specifies the locking behavior associated with the dequeue.
TQueueDeliveryMode	Specifies the type of the message that will be dequeued.
TQueueMessageGrouping	Specifies the message grouping behavior in the queues based on this table.
TQueueMessageState	Specifies the message state.
TQueueNavigation	Specifies the position of the message that will be retrieved.
TQueueSequenceDeviation	Specifies if a message should be enqueued before other messages.
TQueueSortList	Specifies the columnt that will be used as a sort key.
TQueueType	Specifies whether the queue being created is an exception queue or a normal queue.
TQueueVisibility	Specifies the transaction behaviour of the dequeue or enqueue request.

Request Support DAC Forum Provide Feedback

5.16.3.1 TDequeueMode Enumeration

Specifies the locking behavior associated with the dequeue.

Unit

OraAQ

Syntax

```
TDequeueMode = (dqmBrowse, dqmLocked, dqmRemove, dqmRemoveNoData);
```

Values

Value	Meaning
dqmBrowse	Messages will be dequeued without any locking.
dqmLocked	Sets write locks on the dequeued messages. The locks last for the durations of the transaction.
dqmRemove	Messages are removed after dequeuing. The default value.
dqmRemoveNoData	Messages are marked as updated or deleted after dequeuing. The message can be retained in the queue table based on the retention properties.

Remarks

Use DequeueMode property to specify the locking behavior associated with the dequeue.

Request Support DAC Forum Provide Feedback

5.16.3.2 TQueueDeliveryMode Enumeration

Specifies the type of the message that will be dequeued.

Unit

OraAQ

Syntax

```
TQueueDeliveryMode = (qdmPersistent, qdmBuffered,
qdmPersistentOrBuffered);
```

Values

Value	Meaning
qdmBuffered	Only buffered message will be dequeued.
qdmPersistent	Only persistent message will be dequeued.
qdmPersistentOrBuffered	Suited message of any type can be dequeued. Is not used when enqueueing a message.

 Request Support DAC Forum Provide Feedback

5.16.3.3 TQueueMessageGrouping Enumeration

Specifies the message grouping behavior in the queues based on this table.

Unit

OraAQ

Syntax

```
TQueueMessageGrouping = (qmgNone, qmgTransactional);
```

Values

Value	Meaning
qmgNone	Each queue message is treated individually. The default value.
qmgTransactional	Messages, enqueued in one transaction, will belong to the same group.

 Request Support DAC Forum Provide Feedback

5.16.3.4 TQueueMessageState Enumeration

Specifies the message state.

Unit

OraAQ

Syntax

```
TQueueMessageState = (qmsReady, qmsWaiting, qmsProcessed,
qmsExpired);
```

Values

Value	Meaning
qmsExpired	The message has been moved to the exception queue.
qmsProcessed	The message has been processed and is retained.

qmsReady	The message is ready to be processed.
qmsWaiting	The message delay has not yet been reached.

Request Support DAC Forum Provide Feedback

5.16.3.5 TQueueNavigation Enumeration

Specifies the position of the message that will be retrieved.

Unit

OraAQ

Syntax

```
TQueueNavigation = (qnNextMessage, qnNextTransaction,
qnFirstMessage, qnFirstMessageMultiGroup, qnNextMessageMultiGroup);
```

Values

Value	Meaning
qnFirstMessage	The first message which is available and that matches the search criteria will be dequeued. This setting resets the position to the beginning of the queue.
qnFirstMessageMultiGroup	Available matching messages from the beginning of the queue will be dequeued (possibly across different transaction groups) until reaching the limit of result message array size. This setting resets the position to the beginning of the queue.
qnNextMessage	The next message that is available and that matches the search criteria will be dequeued. The default value.
qnNextMessageMultiGroup	The next set of available matching messages from the beginning of the queue will be dequeued (possibly across different transaction groups) until reaching the limit of the result message array size. This setting resets the position to the beginning of the queue.
qnNextTransaction	The first message of the next transaction group that is available and that matches the search criteria will be dequeued.

Request Support DAC Forum Provide Feedback

5.16.3.6 TQueueSequenceDeviation Enumeration

Specifies if a message should be enqueued before other messages.

Unit

OraAQ

Syntax

```
TQueueSequenceDeviation = (qsdNone, qsdBefore, qsdTop);
```

Values

Value	Meaning
qsdBefore	Message is enqueued before the message specified by RelativeMsgid property.
qsdNone	Message is enqueued in usual order. The default value.
qsdTop	The message will be enqueued before all queued messages.

Request Support DAC Forum Provide Feedback

5.16.3.7 TQueueSortList Enumeration

Specifies the columnt that will be used as a sort key.

Unit

OraAQ

Syntax

```
TQueueSortList = (qslDefault, qslPriority, qslEnqueueTime,
qslPriorityEnqueueTime, qslEnqueueTimePriority);
```

Values

Value	Meaning
qslDefault	No sorting is specified, the table will be sorted in the enqueue time in the ascending order by default. The default value.
qslEnqueueTime	The table will be sorted by enq_time column.
qslEnqueueTimePr	The table will be sorted by both enq_time and priority columns.

iority	Enq_time column defines the most significant order.
qslPriority	The table will be sorted by priority column.
qslPriorityEnqueueTime	The table will be sorted by both the enq_time and priority columns. Priority column defines the most significant order.

Request Support DAC Forum Provide Feedback

5.16.3.8 TQueueType Enumeration

Specifies whether the queue being created is an exception queue or a normal queue.

Unit

OraAQ

Syntax

```
TQueueType = (qtNormalQueue, qtExceptionQueue);
```

Values

Value	Meaning
qtExceptionQueue	The queue being created is an exception queue.
qtNormalQueue	The queue being created is a normal queue. The Default value.

Request Support DAC Forum Provide Feedback

5.16.3.9 TQueueVisibility Enumeration

Specifies the transaction behaviour of the dequeue or enqueue request.

Unit

OraAQ

Syntax

```
TQueueVisibility = (qvOnCommit, qvImmediate);
```

Values

Value	Meaning
qvImmediate	The dequeue or enqueue is being made in an autonomous transaction.
qvOnCommit	The dequeue or enqueue is a part of the current transaction. The operation is complete when the transaction commits. The default value.

Request Support DAC Forum Provide Feedback

5.17 OraCall

Defines Oracle Call Interface routines.

Classes

Name	Description
TOracleHome	A class representing the TOraSession.Home property for using Oracle Client.

Request Support DAC Forum Provide Feedback

5.17.1 Classes

Classes in the **OraCall** unit.

Classes

Name	Description
TOracleHome	A class representing the TOraSession.Home property for using Oracle Client.

Request Support DAC Forum Provide Feedback

5.17.1.1 TOracleHome Class

A class representing the TOraSession.Home property for using Oracle Client.

For a list of all members of this type, see TOracleHome members.

Unit

OraCall

Syntax

```
TOracleHome = class(System.TObject);
```

Remarks

Use TOraSession.Home property to select which Oracle client will be used in your application. Use this property in cases when there is a number of Oracle clients on the machine. ODAC searches all available homes in the HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\ALL_HOMES registry folder.

See Also

- TOraSession.Home

Request Support DAC Forum Provide Feedback

5.17.1.1.1 Members

TOracleHome class overview.

Properties

Name	Description
Name	Defines the name of Oracle Client - ORACLE_HOME_NAME
OCICallStyle	Defines belonging to the OCI function set {None, OCI73, OCI80}
OCIClientDLL	Defines the name and full path to the ClientOCI library

OCIDLL	Defines the name and full path to the OCI library
OCIVersion	Defines the version of the OCI library as integer
OCIVersionSt	Defines the version of the OCI library
Path	Defines the path to Oracle Client
PossibleOCICallStyles	Holds the list of supported OCI function sets
TNSPath	Defines the value for the TNS_ADMIN variable

Request Support DAC Forum Provide Feedback

5.17.1.1.2 Properties

Properties of the **TOracleHome** class.

For a complete list of the **TOracleHome** class members, see the TOracleHome Members topic.

Public

Name	Description
Name	Defines the name of Oracle Client - ORACLE_HOME_NAME
OCICallStyle	Defines belonging to the OCI function set {None, OCI73, OCI80}
OCIClientDLL	Defines the name and full path to the ClientOCI library
OCIDLL	Defines the name and full path to the OCI library
OCIVersion	Defines the version of the OCI library as integer
OCIVersionSt	Defines the version of the OCI library
Path	Defines the path to Oracle Client

PossibleOCICallStyles	Holds the list of supported OCI function sets
TNSPath	Defines the value for the TNS_ADMIN variable

See Also

- TOracleHome Class
- TOracleHome Class Members

5.17.1.1.2.1 Name Property

Defines the name of Oracle Client - ORACLE_HOME_NAME

Class

TOracleHome

Syntax

```
property Name: string;
```


5.17.1.1.2.2 OCICallStyle Property

Defines belonging to the OCI function set {None, OCI73, OCI80}

Class

TOracleHome

Syntax

```
property OCICallStyle: TOCICallStyle;
```

5.17.1.1.2.3 OCIClientDLL Property

Defines the name and full path to the ClientOCI library

Class

TOracleHome

Syntax

```
property OCIClientDLL: string;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.4 OCIDLL Property

Defines the name and full path to the OCI library

Class

TOracleHome

Syntax

```
property OCIDLL: string;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.5 OCIVersion Property

Defines the version of the OCI library as integer

Class

TOracleHome

Syntax

```
property OCIVersion: word;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.6 OCIVersionSt Property

Defines the version of the OCI library

Class

TOracleHome

Syntax

```
property OCIVersionSt: string;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.7 Path Property

Defines the path to Oracle Client

Class

TOracleHome

Syntax

```
property Path: string;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.8 PossibleOCICallStyles Property

Holds the list of supported OCI function sets

Class

TOracleHome

Syntax

```
property PossibleOCICallStyles: TOCICallStyleSet;
```

 Request Support DAC Forum Provide Feedback

5.17.1.1.2.9 TNSPath Property

Defines the value for the TNS_ADMIN variable

Class

TOracleHome

Syntax

```
property TNSPath: string;
```


Request Support DAC Forum Provide Feedback

5.18 OraClasses

OraClasses unit defines following data type constants: dtRowId dtCursor dtOraBlob dtOraClob dtBFILE dtCFILE dtLabel dtFixedChar dtUndefined dtTimeStamp dtTimeStampTZ dtTimeStampLTZ dtIntervalYM dtIntervalDS // obsolete dtBLOBLocator = dtOraBlob dtCLOBLocator = dtOraClob

Classes

Name	Description
TNotifyTableChanges	Allows getting the list of tables and changes made to them using the TNotifyTableChanges.Changes property.
TOraCursor	A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCIStmt descriptor for Oracle 8.
TOraFile	A class holding the value of the BFile field and parameter.
TOraInterval	A class providing support for Oracle 9 interval datatypes.
TOraLob	A class holding the value of the BLOB and CLOB fields and parameters.

TOraNumber	A class supporting
TOraTimeStamp	A class supporting Oracle 9 timestamp datatypes.

Enumerations

Name	Description
TChangeNotifyEventType	Contains the event type.
TConnectMode	Specifies the system privileges used when a user connects to the server.
TMessageType	Defines the type of the next message found in the received named pipe local buffer.
TOptimizerMode	Specifies the optimizer mode for connection.

Variables

Name	Description
FloatPrecision	Set this constant to define the type of NUMBER fields with Scale > 0.
IntegerPrecision	Set this constant to define the type of NUMBER fields with precision less than or equal to IntegerPrecision as dtInteger.
LargeIntPrecision	Set this constant to define the type of NUMBER fields with precision greater than the IntegerPresision and less than or equal to LargeIntPrecision as dtLargeInt.

5.18.1 Classes

Classes in the **OraClasses** unit.

Classes

Name	Description
TNotifyTableChanges	Allows getting the list of tables and changes made to them using the TNotifyTableChanges.Changes property.
TOraCursor	A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCIStmt descriptor for Oracle 8.
TOraFile	A class holding the value of the BFile field and parameter.
TOraInterval	A class providing support for Oracle 9 interval datatypes.
TOraLob	A class holding the value of the BLOB and CLOB fields and parameters.
TOraNumber	A class supporting
TOraTimeStamp	A class supporting Oracle 9 timestamp datatypes.

 Request Support DAC Forum Provide Feedback

5.18.1.1 TNotifyTableChanges Class

Allows getting the list of tables and changes made to them using the TNotifyTableChanges.Changes property.

For a list of all members of this type, see TNotifyTableChanges members.

Unit

OraClasses

Syntax

```
TNotifyTableChanges = class(TCustomNotifyChanges);
```

Inheritance Hierarchy

TCustomNotifyChanges

TNotifyTableChanges

Request Support DAC Forum Provide Feedback

5.18.1.1.1 Members

TNotifyTableChanges class overview.

Properties

Name	Description
Changes	Contains the list of tables and changes made to them.

© 1997-2024 Devart. All Rights Reserved.
Request Support DAC Forum Provide Feedback

5.18.1.1.2 Properties

Properties of the **TNotifyTableChanges** class.

For a complete list of the **TNotifyTableChanges** class members, see the TNotifyTableChanges Members topic.

Public

Name	Description
Changes	Contains the list of tables and changes made to them.

See Also

- TNotifyTableChanges Class
- TNotifyTableChanges Class Members

5.18.1.1.2.1 Changes Property(Indexer)

Contains the list of tables and changes made to them.

Class

TNotifyTableChanges

Syntax

```
property Changes[Index: integer]: TNotifyTableChange; default;
```

Parameters

Index

5.18.1.2 TOraCursor Class

A class holding the internal Oracle Call Interface data CDA for Oracle 7 and the OCIStmt descriptor for Oracle 8.

For a list of all members of this type, see TOraCursor members.

Unit

OraClasses

Syntax

```
TOraCursor = class(TCRCursor);
```

Remarks

TOraCursor holds the internal Oracle Call Interface data CDA for Oracle 7 and the OCIStmt descriptor for Oracle 8.

Inheritance Hierarchy

TSharedObject

TCRCursor

TOraCursor

See Also

- TCursorField
- TOraDataSet.Cursor
- TOraParam.AsCursor

Request Support DAC Forum Provide Feedback

5.18.1.2.1 Members

TOraCursor class overview.

Properties

Name	Description
CDA	Used to return CDA.
OCICallStyle	Used to get or set the Oracle Client version for a subsequent cursor operations.
OCIStmt	Used to return the OCIStmt handler.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
State	Used to set the cursor state.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocCursor	Allocates CDA for Oracle 7 and OCIStmt handler for Oracle 8.

CanFetch	Verifies if the cursor state permits data fetching.
FreeCursor	Releases the cursor data.
Release (inherited from **TSharedObject**)	Decrements the reference count.

 Request Support DAC Forum Provide Feedback

5.18.1.2.2 Properties

Properties of the **TOraCursor** class.

For a complete list of the **TOraCursor** class members, see the TOraCursor Members topic.

Public

Name	Description
CDA	Used to return CDA.
OCICallStyle	Used to get or set the Oracle Client version for a subsequent cursor operations.
OCIStmt	Used to return the OCIStmt handler.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
State	Used to set the cursor state.

See Also

- TOraCursor Class
- TOraCursor Class Members

 Request Support DAC Forum Provide Feedback

5.18.1.2.2.1 CDA Property

Used to return CDA.

Class

TOraCursor

Syntax

```
property CDA: PCDA;
```

Remarks

Use the CDA property to return CDA. You can use this property for Oracle 7.

 Request Support DAC Forum Provide Feedback

5.18.1.2.2.2 OCICallStyle Property

Used to get or set the Oracle Client version for a subsequent cursor operations.

Class

TOraCursor

Syntax

```
property OCICallStyle: TOCICallStyle;
```

Remarks

Use the OCICallStyle property to get or set the Oracle Client version for a subsequent cursor operations.

 Request Support DAC Forum Provide Feedback

5.18.1.2.2.3 OCIStmt Property

Used to return the OCIStmt handler.

Class

TOraCursor

Syntax

```
property OCIStmt: pOCIStmt;
```

Remarks

Use the OCIStmt property to return the OCIStmt handler. You can use this property for Oracle 8.

 Request Support DAC Forum Provide Feedback

5.18.1.2.2.4 State Property

Used to set the cursor state.

Class

TOraCursor

Syntax

```
property State: TCursorState;
```

See Also

- TCursorState

 Request Support DAC Forum Provide Feedback

5.18.1.2.3 Methods

Methods of the **TOraCursor** class.

For a complete list of the **TOraCursor** class members, see the TOraCursor Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of

	references dependent on the TSharedObject object.
AllocCursor	Allocates CDA for Oracle 7 and OCIStmt handler for Oracle 8.
CanFetch	Verifies if the cursor state permits data fetching.
FreeCursor	Releases the cursor data.
Release (inherited from **TSharedObject**)	Decrements the reference count.

See Also

- TOraCursor Class
- TOraCursor Class Members

Request Support DAC Forum Provide Feedback

5.18.1.2.3.1 AllocCursor Method

Allocates CDA for Oracle 7 and OCIStmt handler for Oracle 8.

Class

TOraCursor

Syntax

```
procedure AllocCursor(StatementMode: TStatementMode =
smAllocated);
```

Parameters

AStatementCache

Remarks

Call the AllocCursor procedure to allocate CDA for Oracle 7 and OCIStmt handler for Oracle 8.

See Also

- FreeCursor

5.18.1.2.3.2 CanFetch Method

Verifies if the cursor state permits data fetching.

Class

TOraCursor

Syntax

```
function CanFetch: boolean; override;
```

Return Value

True, if data fetching is permitted, False otherwise.

Remarks

Call the CanFetch function to verify whether the cursor is in the state that permits data fetching.

5.18.1.2.3.3 FreeCursor Method

Releases the cursor data.

Class

TOraCursor

Syntax

```
procedure FreeCursor;
```

Remarks

Call the FreeCursor method to release the cursor data.

See Also

- AllocCursor

5.18.1.3 TOraFile Class

A class holding the value of the BFile field and parameter.

For a list of all members of this type, see TOraFile members.

Unit

OraClasses

Syntax

```
TOraFile = class(TOraLob);
```

Remarks

TOraFile is a descendant of the TOraLob class. It holds the value of the BFile field and parameter.

The BFile datatype provides access to the file LOBs that are stored in file systems outside an Oracle database. Oracle 8 currently supports access to binary files, or BFILEs. The BFILE datatype allows the read-only support of large binary files; you cannot modify a file through Oracle.

Inheritance Hierarchy

TSharedObject
  TBlob
    TCompressedBlob
      TOraLob
        TOraFile

See Also

- TOraLob
- TBFileField
- TOraParam.AsBFile

5.18.1.3.1 Members

TOraFile class overview.

Properties

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Cached (inherited from **TOraLob**)	Used to indicate where the LOB data is.
Compressed (inherited from **TCompressedBlob**)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from **TCompressedBlob**)	Used to indicate compressed size of the Blob data.
FileDir	Determines the directory alias of where the file associated with the BFile field is stored.
FileName	Used to determine the name of a file associated with the BFile field.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
OCILobLocator (inherited from **TOraLob**)	Used to get or set the OCILobLocator handle.
OCILobLocatorPtr (inherited from **TOraLob**)	Used to retrieve the LOB value type locator.
OCISvcCtx (inherited from **TOraLob**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocLob (inherited from **TOraLob**)	Allocates and initializes the LOB locator.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
Close	Closes a file opened by Open.
CreateTemporary (inherited from **TOraLob**)	Allocates and initializes a temporary LOB locator of the specified type.
DisableBuffering (inherited from **TOraLob**)	Disables the LOB buffering for the LOB locator.
EnableBuffering (inherited from **TOraLob**)	Enables the LOB buffering for the LOB locator.
Exists	Determines whether a file associated with the BFile field exists.
FreeLob (inherited from **TOraLob**)	Releases the LOB locator descriptor.
FreeTemporary (inherited from **TOraLob**)	Frees a temporary LOB.
Init (inherited from **TOraLob**)	Initializes the OCILobLocator handle.
IsInit (inherited from **TOraLob**)	Verifies if the LOB locator is initialized.
IsOpen	Determines whether the file associated with the BFile field is opened.
IsTemporary (inherited from **TOraLob**)	Indicates whether the LOB is temporary.
LengthLob (inherited from **TOraLob**)	Returns the number of bytes contained in the LOB object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.

Open	Opens the file associated with the BFile field.
Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadLob (inherited from **TOraLob**)	Reads the LOB content from the server.
Refresh	Reloads a file associated the BFile field opened by Open.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.
WriteLob (inherited from **TOraLob**)	Writes a LOB value to the server.

Request Support DAC Forum Provide Feedback

5.18.1.3.2 Properties

Properties of the **TOraFile** class.

For a complete list of the **TOraFile** class members, see the TOraFile Members topic.

Public

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Cached (inherited from **TOraLob**)	Used to indicate where the LOB data is.
Compressed (inherited from **TCompressedBlob**)	Used to indicate if the Blob is compressed.

CompressedSize (inherited from **TCompressedBlob**)	Used to indicate compressed size of the Blob data.
FileDir	Determines the directory alias of where the file associated with the BFile field is stored.
FileName	Used to determine the name of a file associated with the BFile field.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
OCILobLocator (inherited from **TOraLob**)	Used to get or set the OCILobLocator handle.
OCILobLocatorPtr (inherited from **TOraLob**)	Used to retrieve the LOB value type locator.
OCISvcCtx (inherited from **TOraLob**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

See Also

- TOraFile Class
- TOraFile Class Members

Request Support DAC Forum Provide Feedback

5.18.1.3.2.1 FileDir Property

Determines the directory alias of where the file associated with the BFile field is stored.

Class

`TOraFile`

Syntax

```
property FileDir: string;
```

Remarks

Use FileDir to determine the directory alias where the file associated with the BFile field is stored.

To create a directory alias use CREATE DIRECTORY.

See Also

- FileName

 Request Support DAC Forum Provide Feedback

5.18.1.3.2.2 FileName Property

Used to determine the name of a file associated with the BFile field.

Class

TOraFile

Syntax

```
property FileName: string;
```

Remarks

Use the FileName property to determine the name of a file associated with the BFile field.

See Also

- FileDir

 Request Support DAC Forum Provide Feedback

5.18.1.3.3 Methods

Methods of the **TOraFile** class.

For a complete list of the **TOraFile** class members, see the TOraFile Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocLob (inherited from **TOraLob**)	Allocates and initializes the LOB locator.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
Close	Closes a file opened by Open.
CreateTemporary (inherited from **TOraLob**)	Allocates and initializes a temporary LOB locator of the specified type.
DisableBuffering (inherited from **TOraLob**)	Disables the LOB buffering for the LOB locator.
EnableBuffering (inherited from **TOraLob**)	Enables the LOB buffering for the LOB locator.
Exists	Determines whether a file associated with the BFile field exists.
FreeLob (inherited from **TOraLob**)	Releases the LOB locator descriptor.
FreeTemporary (inherited from **TOraLob**)	Frees a temporary LOB.
Init (inherited from **TOraLob**)	Initializes the OCILobLocator handle.
IsInit (inherited from **TOraLob**)	Verifies if the LOB locator is initialized.
IsOpen	Determines whether the file associated with the BFile field is opened.
IsTemporary (inherited from **TOraLob**)	Indicates whether the LOB is temporary.
LengthLob (inherited from **TOraLob**)	Returns the number of bytes contained in the LOB object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.

Open	Opens the file associated with the BFile field.
Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadLob (inherited from **TOraLob**)	Reads the LOB content from the server.
Refresh	Reloads a file associated the BFile field opened by Open.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.
WriteLob (inherited from **TOraLob**)	Writes a LOB value to the server.

See Also

- TOraFile Class
- TOraFile Class Members

Request Support DAC Forum Provide Feedback

5.18.1.3.3.1 Close Method

Closes a file opened by Open.

Class

TOraFile

Syntax

```
procedure Close;
```

Remarks

Call the Close procedure to close a file specified by FileDir and FileName properties of the BFile field, opened earlier by Open.

See Also

- Open

Request Support DAC Forum Provide Feedback

5.18.1.3.3.2 Exists Method

Determines whether a file associated with the BFile field exists.

Class

TOraFile

Syntax

```
function Exists: boolean;
```

Return Value

True, if a file associated with the BFile field exists, False otherwise.

Remarks

Call the Exists method to determine if a file associated with the BFile field exists.

See Also

- Open

Request Support DAC Forum Provide Feedback

5.18.1.3.3.3 IsOpen Method

Determines whether the file associated with the BFile field is opened.

Class

TOraFile

Syntax

```
function IsOpen: boolean;
```

Return Value

True, if the file is opened, False otherwise.

Remarks

Use the IsOpen method to determine whether the file associated with the BFile field is opened.

See Also

- Open

 Request Support DAC Forum Provide Feedback

5.18.1.3.3.4 Open Method

Opens the file associated with the BFile field.

Class

TOraFile

Syntax

```
procedure Open;
```

Remarks

Call the Open procedure to open the file associated with the BFile field.

See Also

- Close
- IsOpen
- Exists

 Request Support DAC Forum Provide Feedback

5.18.1.3.3.5 Refresh Method

Reloads a file associated the BFile field opened by Open.

Class

TOraFile

Syntax

```
procedure Refresh;
```

Remarks

Call the Refresh procedure to reload the file associated the BFile field, which was opened earlier by Open.

See Also

- Open

 Request Support DAC Forum Provide Feedback

5.18.1.4 TOraInterval Class

A class providing support for Oracle 9 interval datatypes.

For a list of all members of this type, see TOraInterval members.

Unit

OraClasses

Syntax

```
TOraInterval = class(TSharedObject);
```

Remarks

TOraInterval is used to support Oracle 9 interval datatypes. There are two interval datatypes:

INTERVAL YEAR TO MONTH,

INTERVAL DAY TO SECOND.

They can be distinguished by the DescriptorType property.

Inheritance Hierarchy

TSharedObject

TOraInterval

5.18.1.4.1 Members

TOraInterval class overview.

Properties

Name	Description
AsString	Used to get and set the interval value as string.
DescriptorType	Used to define the descriptor type allocated to operate with the interval value.
FracPrecision	Used to get or set the interval fractional second precision.
IsNull	Used to get or set the null interval value flag.
LeadPrecision	Used to get or set the leading field precision.
OCIInterval	Used to get or set the OCIInterval descriptor handle.
OCIIntervalPtr	Used to get the OCIInterval type locator.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description

AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocInterval	Allocates the OCIInterval descriptor handle.
Compare	Compares the current TOraInterval value with the Dest value.
FreeInterval	Frees the OCIInterval descriptor handle.
GetDaySecond	Gets the values of the day and second from an interval.
GetYearMonth	Gets the values of the year and month from an interval.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SetDaySecond	Used to set the values of the day and second from in an interval.
SetYearMonth	Sets the values of the year and month in an interval.

5.18.1.4.2 Properties

Properties of the **TOraInterval** class.

For a complete list of the **TOraInterval** class members, see the TOraInterval Members topic.

Public

Name	Description
AsString	Used to get and set the interval value as string.
DescriptorType	Used to define the descriptor type allocated to operate with the interval value.
FracPrecision	Used to get or set the interval fractional second precision.

IsNull	Used to get or set the null interval value flag.
LeadPrecision	Used to get or set the leading field precision.
OCIInterval	Used to get or set the OCIInterval descriptor handle.
OCIIntervalPtr	Used to get the OCIInterval type locator.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

See Also

- TOraInterval Class
- TOraInterval Class Members

Request Support DAC Forum Provide Feedback

5.18.1.4.2.1 AsString Property

Used to get and set the interval value as string.

Class

TOraInterval

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the interval value as string. Oracle establishes certain formats for each interval type.

INTERVAL YEAR TO MONTH format:

'[year-]month'.

INTERVAL DAY TO SECOND formats:

'seconds',

'minutes[:seconds]',

'hours[:minutes:[seconds]]',

'days[hours[:minutes:[seconds]]]'.

Optional fields are surrounded by brackets.

Reading AsString property when IsNull is True returns an empty string.

See Also

- IsNull

 Request Support DAC Forum Provide Feedback

5.18.1.4.2.2 DescriptorType Property

Used to define the descriptor type allocated to operate with the interval value.

Class

TOraInterval

Syntax

```
property DescriptorType: cardinal;
```

Remarks

Use the DescriptorType property to define the type of the descriptor which is allocated to operate with the interval value. In most cases you don't need to set this property directly: it is adjusted automatically when working with fields and parameters.

Valid values for this property are:

OCI_DTYPE_INTERVAL_DS,

OCI_DTYPE_INTERVAL_YM.

They are defined in OraCall unit.

 Request Support DAC Forum Provide Feedback

5.18.1.4.2.3 FracPrecision Property

Used to get or set the interval fractional second precision.

Class

TOraInterval

Syntax

```
property FracPrecision: byte;
```

Remarks

Use the FracPrecision property to get or set fractional second precision of the interval (the number of digits that are used to represent fractional seconds). This property affects only when reading the AsString property. The default value of the property is 6.

See Also

- AsString

Request Support DAC Forum Provide Feedback

5.18.1.4.2.4 IsNull Property

Used to get or set the null interval value flag.

Class

TOraInterval

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to get or set the null interval value flag. When IsNull is True getting values via the GetDaySecond and GetYearMonth methods can raise an exception. The

AsString property in this case returns an empty string.

See Also

- GetDaySecond
- GetYearMonth
- AsString

 Request Support DAC Forum Provide Feedback

5.18.1.4.2.5 LeadPrecision Property

Used to get or set the leading field precision.

Class

TOraInterval

Syntax

```
property LeadPrecision: byte;
```

Remarks

Use the LeadPrecision property to get or set the leading field precision (the number of digits that are used to represent leading interval part). This property has effect only when reading the AsString property. The default value of the property is 2.

See Also

- AsString

 Request Support DAC Forum Provide Feedback

5.18.1.4.2.6 OCIInterval Property

Used to get or set the OCIInterval descriptor handle.

Class

TOraInterval

Syntax

```
property OCIInterval: pOCIInterval;
```

Remarks

Use the OCIInterval property to get or set the OCIInterval descriptor handle.

Request Support DAC Forum Provide Feedback

5.18.1.4.2.7 OCIIntervalPtr Property

Used to get the OCIInterval type locator.

Class

TOraInterval

Syntax

```
property OCIIntervalPtr: ppOCIInterval;
```

Remarks

Use the OCIIntervalPtr property to get the OCIInterval type locator.

Request Support DAC Forum Provide Feedback

5.18.1.4.3 Methods

Methods of the **TOraInterval** class.

For a complete list of the **TOraInterval** class members, see the TOraInterval Members topic.

Protected

Name	Description
AllocInterval	Allocates the OCIInterval descriptor handle.
FreeInterval	Frees the OCIInterval descriptor handle.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Compare	Compares the current TOraInterval value with the Dest value.
GetDaySecond	Gets the values of the day and second from an interval.
GetYearMonth	Gets the values of the year and month from an interval.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SetDaySecond	Used to set the values of the day and second from in an interval.
SetYearMonth	Sets the values of the year and month in an interval.

See Also

- TOraInterval Class
- TOraInterval Class Members

5.18.1.4.3.1 AllocInterval Method

Allocates the OCIInterval descriptor handle.

Class

TOraInterval

Syntax

```
procedure AllocInterval;
```

Remarks

Call the AllocInterval method to allocate the OCIInterval descriptor handle according to the DescriptorType property.

See Also

- DescriptorType

 Request Support DAC Forum Provide Feedback

5.18.1.4.3.2 Compare Method

Compares the current TOraInterval value with the Dest value.

Class

TOraInterval

Syntax

```
function Compare(Dest: TOraInterval): integer;
```

Parameters

Dest

Holds the Dest value.

Return Value

the negative value if current interval is shorter than Dest, zero if intervals are equal, and positive value if the current interval is longer than Dest.

Remarks

Call the Compare method to compare the current TOraInterval value with Dest value.

Returns negative value if current interval is shorter than Dest, zero if intervals are equal and positive value if current interval is longer than Dest.

 Request Support DAC Forum Provide Feedback

5.18.1.4.3.3 FreeInterval Method

Frees the OCIInterval descriptor handle.

Class

TOraInterval

Syntax

```
procedure FreeInterval;
```

Remarks

Call the FreeInterval procedure frees the OCIInterval descriptor handle. After the FreeInterval call the IsNull property is set to True.

See Also

- IsNull

Request Support DAC Forum Provide Feedback

5.18.1.4.3.4 GetDaySecond Method

Gets the values of the day and second from an interval.

Class

TOraInterval

Syntax

```
procedure GetDaySecond(var Day: integer; var Hour: integer; var
Min: integer; var Sec: integer; var FSec: integer);
```

Parameters

Day
Holds the day value.

Hour
Holds the hour value.

Min
Holds the minute value.

Sec
Holds the second value.

FSec
Holds the fraction second value.

Remarks

Call the GetDaySecond procedure to get the values of the day and second from an interval.

See Also

- SetDaySecond

Request Support DAC Forum Provide Feedback

5.18.1.4.3.5 GetYearMonth Method

Gets the values of the year and month from an interval.

Class

TOraInterval

Syntax

```
procedure GetYearMonth(var Year: integer; var Month: integer);
```

Parameters

Year
Holds the year value.

Month
Holds the month value.

Remarks

Call the GetYearMonth procedure to get the values of the year and month from an interval.

See Also

- SetYearMonth

Request Support DAC Forum Provide Feedback

5.18.1.4.3.6 SetDaySecond Method

Used to set the values of the day and second from in an interval.

Class

TOraInterval

Syntax

```
procedure SetDaySecond(Day: integer; Hour: integer; Min: integer;
Sec: integer; FSec: integer);
```

Parameters

Day
Holds the day value.

Hour
Holds the hour value.

Min
Holds the minute value.

Sec
Holds the second value.

FSec
Holds the fraction second value.

Remarks

Call the SetDaySecond method to set the values of the day and second from in an interval.

See Also

- GetDaySecond

Request Support DAC Forum Provide Feedback

5.18.1.4.3.7 SetYearMonth Method

Sets the values of the year and month in an interval.

Class

TOraInterval

Syntax

```
procedure SetYearMonth(Year: integer; Month: integer);
```

Parameters

Year
Holds the year value.

Month
Holds the month value.

Remarks

Call the SetYearMonth method to set the values of the year and month in an interval.

See Also

- GetYearMonth

Request Support DAC Forum Provide Feedback

5.18.1.5 TOraLob Class

A class holding the value of the BLOB and CLOB fields and parameters.

For a list of all members of this type, see TOraLob members.

Unit

OraClasses

Syntax

```
TOraLob = class(TCompressedBlob);
```

Remarks

TOraLob is a descendant of the TBlob class. It holds value of BLOB and CLOB fields and parameters.

Note: You can affect performance of reading/writing LOBs by changing MemData.DefaultPieceSize variable to a different value. DefaultPieceSize defines the size of data portion transferred through network at a single OCI call.

Inheritance Hierarchy

TSharedObject
  TBlob
    TCompressedBlob
      TOraLob

See Also

- TBlob
- TOraDataSet.GetLob
- TOraParam.AsBLOBLocator

Request Support DAC Forum Provide Feedback

5.18.1.5.1 Members

TOraLob class overview.

Properties

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Cached	Used to indicate where the LOB data is.
Compressed (inherited from **TCompressedBlob**)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from **TCompressedBlob**)	Used to indicate compressed size of the Blob data.
IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
OCILobLocator	Used to get or set the OCILobLocator handle.
OCILobLocatorPtr	Used to retrieve the LOB

	value type locator.
OCISvcCtx	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocLob	Allocates and initializes the LOB locator.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
CreateTemporary	Allocates and initializes a temporary LOB locator of the specified type.
DisableBuffering	Disables the LOB buffering for the LOB locator.
EnableBuffering	Enables the LOB buffering for the LOB locator.
FreeLob	Releases the LOB locator descriptor.
FreeTemporary	Frees a temporary LOB.
Init	Initializes the OCILobLocator handle.
IsInit	Verifies if the LOB locator is initialized.
IsTemporary	Indicates whether the LOB is temporary.
LengthLob	Returns the number of bytes contained in the LOB object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.

LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.
Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadLob	Reads the LOB content from the server.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.
WriteLob	Writes a LOB value to the server.

Request Support DAC Forum Provide Feedback

5.18.1.5.2 Properties

Properties of the **TOraLob** class.

For a complete list of the **TOraLob** class members, see the TOraLob Members topic.

Public

Name	Description
AsString (inherited from **TBlob**)	Used to manipulate BLOB value as string.
AsWideString (inherited from **TBlob**)	Used to manipulate BLOB value as Unicode string.
Cached	Used to indicate where the LOB data is.
Compressed (inherited from **TCompressedBlob**)	Used to indicate if the Blob is compressed.
CompressedSize (inherited from **TCompressedBlob**)	Used to indicate compressed size of the Blob data.

IsUnicode (inherited from **TBlob**)	Gives choice of making TBlob store and process data in Unicode format or not.
OCILobLocator	Used to get or set the OCILobLocator handle.
OCILobLocatorPtr	Used to retrieve the LOB value type locator.
OCISvcCtx	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TBlob**)	Used to learn the size of the TBlob value in bytes.

See Also

- TOraLob Class
- TOraLob Class Members

Request Support DAC Forum Provide Feedback

5.18.1.5.2.1 Cached Property

Used to indicate where the LOB data is.

Class

TOraLob

Syntax

```
property Cached: boolean;
```

Remarks

Use the Cached property to indicate whether the LOB data is cached on a client or it is accessed remotely on the server. In most cases you don't need to set the value of this property directly. To enable or disable LOB caching use the TOraDataSet.Options property.

See Also

- TOraDataSet.Options

 Request Support DAC Forum Provide Feedback

5.18.1.5.2.2 OCILobLocator Property

Used to get or set the OCILobLocator handle.

Class

TOraLob

Syntax

```
property OCILobLocator: pOCILobLocator;
```

Remarks

Use the OCILobLocator property to get or set the OCILobLocator handle.

 Request Support DAC Forum Provide Feedback

5.18.1.5.2.3 OCILobLocatorPtr Property

Used to retrieve the LOB value type locator.

Class

TOraLob

Syntax

```
property OCILobLocatorPtr: ppOCILobLocator;
```

Remarks

Use the OCILobLocatorPtr property to retrieve the LOB value type locator.

 Request Support DAC Forum Provide Feedback

5.18.1.5.2.4 OCISvcCtx Property

Used to assign a service context handle.

Class

TOraLob

Syntax

```
property OCISvcCtx: TOCISvcCtx;
```

Remarks

Use the OCISvcCtx property to assign a service context handle. Some operations with LOBs require service context handle. To get a service context handle use TOraSession.OCISvcCtx.

See Also

- TOraSession.OCISvcCtx

Request Support DAC Forum Provide Feedback

5.18.1.5.3 Methods

Methods of the **TOraLob** class.

For a complete list of the **TOraLob** class members, see the TOraLob Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocLob	Allocates and initializes the LOB locator.
Assign (inherited from **TBlob**)	Sets BLOB value from another TBlob object.
Clear (inherited from **TBlob**)	Deletes the current value in TBlob object.
CreateTemporary	Allocates and initializes a

	temporary LOB locator of the specified type.
DisableBuffering	Disables the LOB buffering for the LOB locator.
EnableBuffering	Enables the LOB buffering for the LOB locator.
FreeLob	Releases the LOB locator descriptor.
FreeTemporary	Frees a temporary LOB.
Init	Initializes the OCILobLocator handle.
IsInit	Verifies if the LOB locator is initialized.
IsTemporary	Indicates whether the LOB is temporary.
LengthLob	Returns the number of bytes contained in the LOB object.
LoadFromFile (inherited from **TBlob**)	Loads the contents of a file into a TBlob object.
LoadFromStream (inherited from **TBlob**)	Copies the contents of a stream into the TBlob object.
Read (inherited from **TBlob**)	Acquires a raw sequence of bytes from the data stored in TBlob.
ReadLob	Reads the LOB content from the server.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToFile (inherited from **TBlob**)	Saves the contents of the TBlob object to a file.
SaveToStream (inherited from **TBlob**)	Copies the contents of a TBlob object to a stream.
Truncate (inherited from **TBlob**)	Sets new TBlob size and discards all data over it.
Write (inherited from **TBlob**)	Stores a raw sequence of bytes into a TBlob object.
WriteLob	Writes a LOB value to the server.

See Also

- TOraLob Class

- TOraLob Class Members

 Request Support DAC Forum Provide Feedback

5.18.1.5.3.1 AllocLob Method

Allocates and initializes the LOB locator.

Class

TOraLob

Syntax

```
procedure AllocLob; virtual;
```

Remarks

Call the AllocLob method to allocate and initialize the LOB locator.

 Request Support DAC Forum Provide Feedback

5.18.1.5.3.2 CreateTemporary Method

Allocates and initializes a temporary LOB locator of the specified type.

Class

TOraLob

Syntax

```
procedure CreateTemporary(LobType: TLobType);
```

Parameters

LobType

Holds the type of the temporary LOB locator.

Remarks

Call the CreateTemporary method to allocate and initialize a temporary LOB locator of the specified type.

5.18.1.5.3.3 DisableBuffering Method

Disables the LOB buffering for the LOB locator.

Class

TOraLob

Syntax

```
procedure DisableBuffering;
```

Remarks

Call the DisableBuffering method to disable the LOB buffering for the LOB locator. The next time data is read from or written to the LOB through the input locator, the LOB buffering subsystem is not used.

See Also

- EnableBuffering

5.18.1.5.3.4 EnableBuffering Method

Enables the LOB buffering for the LOB locator.

Class

TOraLob

Syntax

```
procedure EnableBuffering;
```

Remarks

Call the EnableBuffering method to enable the LOB buffering for the LOB locator. The next time data is read from or written to the LOB through the input locator, the LOB buffering

subsystem is used.

See Also

- DisableBuffering

5.18.1.5.3.5 FreeLob Method

Releases the LOB locator descriptor.

Class

TOraLob

Syntax

```
procedure FreeLob;
```

Remarks

Call the FreeLob method to release the LOB locator descriptor.

5.18.1.5.3.6 FreeTemporary Method

Frees a temporary LOB.

Class

TOraLob

Syntax

```
procedure FreeTemporary;
```

Remarks

Use the FreeTemporary method to free a temporary LOB.

See Also

- FreeLob

 Request Support DAC Forum Provide Feedback

5.18.1.5.3.7 Init Method

Initializes the OCILobLocator handle.

Class

TOraLob

Syntax

```
procedure Init;
```

Remarks

Call the Init method to initialize the OCILobLocator handle.

 Request Support DAC Forum Provide Feedback

5.18.1.5.3.8 IsInit Method

Verifies if the LOB locator is initialized.

Class

TOraLob

Syntax

```
function IsInit: boolean;
```

Return Value

True, if the LOB locator is initialized. False otherwise.

Remarks

Call the IsInit method to verify that the LOB locator is initialized.

 Request Support DAC Forum Provide Feedback

5.18.1.5.3.9 IsTemporary Method

Indicates whether the LOB is temporary.

Class

TOraLob

Syntax

```
function IsTemporary: LongBool;
```

Return Value

True, if the LOB is temporary. False otherwise.

Remarks

Call the IsTemporary method to indicate whether the LOB is temporary.

Request Support DAC Forum Provide Feedback

5.18.1.5.3.10 LengthLob Method

Returns the number of bytes contained in the LOB object.

Class

TOraLob

Syntax

```
function LengthLob: Cardinal;
```

Return Value

the LOB object size in bytes.

Remarks

Call the LengthLob method to return the number of bytes contained in the LOB object.

Request Support DAC Forum Provide Feedback

5.18.1.5.3.11 ReadLob Method

Reads the LOB content from the server.

Class

TOraLob

Syntax

```
procedure ReadLob(var SharedPiece: PPieceHeader; PrefetchLobSize:
Integer); overload;procedure ReadLob; overload;
```

Remarks

Call the ReadLob method to get the LOB content from the server. When reading such properties as AsString or AsWideString this method is called automatically.

Request Support DAC Forum Provide Feedback

5.18.1.5.3.12 WriteLob Method

Writes a LOB value to the server.

Class

TOraLob

Syntax

```
procedure WriteLob;
```

Remarks

Call the WriteLob method to write a LOB value to the server.

Request Support DAC Forum Provide Feedback

5.18.1.6 TOraNumber Class

A class supporting

For a list of all members of this type, see TOraNumber members.

Unit

OraClasses

Syntax

```
TOraNumber = class(TSharedObject);
```

Remarks

TOraNumber is used to support the Oracle numbers in native format. Support of the internal Oracle number format on clients allows to use full number precision without accuracy losses.

Inheritance Hierarchy

TSharedObject

TOraNumber

Request Support DAC Forum Provide Feedback

5.18.1.6.1 Members

TOraNumber class overview.

Properties

Name	Description
AsFloat	Used to get and set the number value as float.
AsInteger	Used to get and set the number value as integer.
AsLargeInt	Used to get and set the number value as Int64.
AsString	Used to get and set the number value as string.
IsNull	Used to set the null number value flag.
OCINumber	Used to get or set the OCINumber opaque structure value.
OCINumberPtr	Used to get a pointer to the OCINumber structure.

RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AssignTo	Copies all properties of the current TOraNumber instance to the Dest TOraNumber instance.
Compare	Compares the current TOraNumber value with the Dest value.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support DAC Forum Provide Feedback

5.18.1.6.2 Properties

Properties of the **TOraNumber** class.

For a complete list of the **TOraNumber** class members, see the TOraNumber Members topic.

Public

Name	Description
AsFloat	Used to get and set the number value as float.
AsInteger	Used to get and set the number value as integer.
AsLargeInt	Used to get and set the number value as Int64.
AsString	Used to get and set the number value as string.

IsNull	Used to set the null number value flag.
OCINumber	Used to get or set the OCINumber opaque structure value.
OCINumberPtr	Used to get a pointer to the OCINumber structure.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

See Also

- TOraNumber Class
- TOraNumber Class Members

Request Support DAC Forum Provide Feedback

5.18.1.6.2.1 AsFloat Property

Used to get and set the number value as float.

Class

TOraNumber

Syntax

```
property AsFloat: double;
```

Remarks

Use the AsFloat property to get and set the number value as float.

Reading AsFloat property when IsNull is True returns 0.

See Also

- IsNull

Request Support DAC Forum Provide Feedback

5.18.1.6.2.2 AsInteger Property

Used to get and set the number value as integer.

Class

TOraNumber

Syntax

```
property AsInteger: Integer;
```

Remarks

Use the AsIntegert property to get and set the number value as integer.

Reading AsInteger property when IsNull is True returns 0.

See Also

- IsNull

Request Support DAC Forum Provide Feedback

5.18.1.6.2.3 AsLargeInt Property

Used to get and set the number value as Int64.

Class

TOraNumber

Syntax

```
property AsLargeInt: Int64;
```

Remarks

Use the AsLargeInt property to get and set the number value as Int64.

Reading AsLargeInt property when IsNull is True returns 0.

See Also

- IsNull

5.18.1.6.2.4 AsString Property

Used to get and set the number value as string.

Class

TOraNumber

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the number value as string.

Reading AsString property when IsNull is True returns empty string.

See Also

- IsNull

5.18.1.6.2.5 IsNull Property

Used to set the null number value flag.

Class

TOraNumber

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to get or set the null number value flag.

Reserved.

5.18.1.6.2.6 OCINumber Property

Used to get or set the OCINumber opaque structure value.

Class

TOraNumber

Syntax

```
property OCINumber: OCINumber;
```

Remarks

Use the OCINumber property to get or set the OCINumber opaque structure value.

Request Support DAC Forum Provide Feedback

5.18.1.6.2.7 OCINumberPtr Property

Used to get a pointer to the OCINumber structure.

Class

TOraNumber

Syntax

```
property OCINumberPtr: pOCINumber;
```

Remarks

Use the OCINumberPtr property to get a pointer to the OCINumber structure.

See Also

- OCINumber

Request Support DAC Forum Provide Feedback

5.18.1.6.3 Methods

Methods of the **TOraNumber** class.

For a complete list of the **TOraNumber** class members, see the TOraNumber Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AssignTo	Copies all properties of the current TOraNumber instance to the Dest TOraNumber instance.
Compare	Compares the current TOraNumber value with the Dest value.
Release (inherited from **TSharedObject**)	Decrements the reference count.

See Also

- TOraNumber Class
- TOraNumber Class Members

5.18.1.6.3.1 AssignTo Method

Copies all properties of the current TOraNumber instance to the Dest TOraNumber instance.

Class

TOraNumber

Syntax

```
procedure AssignTo(Dest: TOraNumber);
```

Parameters

Dest

Holds a Dest TOraNumber instance.

Remarks

Call AssignTo method to copy all properties of the current TOraNumber instance to the Dest TOraNumber instance.

 Request Support DAC Forum Provide Feedback

5.18.1.6.3.2 Compare Method

Compares the current TOraNumber value with the Dest value.

Class

TOraNumber

Syntax

```
function Compare(Dest: TOraNumber): integer;
```

Parameters

Dest

Holds the Dest value to compare the current TOraNumber value with.

Return Value

a negative value if current number is less than Dest, zero if numbers are equal, and a positive value if the current number is greater than Dest.

Remarks

Call the Compare method to compare the current TOraNumber value with the Dest value.

Returns a negative value if current number is less than Dest, zero if numbers are equal, and a positive value if the current number is greater than Dest.

 Request Support DAC Forum Provide Feedback

5.18.1.7 TOraTimeStamp Class

A class supporting Oracle 9 timestamp datatypes.

For a list of all members of this type, see TOraTimeStamp members.

Unit

OraClasses

Syntax

```
TOraTimeStamp = class(TSharedObject);
```

Remarks

TOraTimeStamp is used to support Oracle 9 timestamp datatypes. There are three timestamp datatypes:

TIMESTAMP,

TIMESTAMP WITH TIME ZONE,

TIMESTAMP WITH LOCAL TIME ZONE.

They can be distinguished by DescriptorType property.

Inheritance Hierarchy

TSharedObject

TOraTimeStamp

Request Support DAC Forum Provide Feedback

5.18.1.7.1 Members

TOraTimeStamp class overview.

Properties

Name	Description
AsDateTime	Used to get and set the timestamp value as TDateTime.

AsString	Used to get and set the timestamp value as string.
DescriptorType	Defines the type of the descriptor allocated to operate with the interval value.
Format	Used to get or set the date-time format model for operations with the TOraTimeStamp.AsString property.
IsNull	Used to get or set the null timestamp value flag.
OCIDateTime	Used to set the OCIDateTime descriptor handle.
OCIDateTimePtr	Used to provide the OCIDateTime type locator.
Precision	Used to learn or set the precision with which the fractional seconds are returned.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
TimeZone	Used to provide a time zone name portion of the datetime value.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocDateTime	Allocates the OCIDateTime descriptor handle according to the TOraTimeStamp.DescriptorType property.
Compare	Compares the current TOraTimeStamp value with the Dest value.

Construct	Constructs datetime descriptor.
GetDate	Provides a date portion of the datetime value.
GetTime	Provides a time portion of the datetime value.
GetTimeZoneOffset	Provides a time zone portion of the datetime value.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SetDate	Provides a date portion of the datetime value.
SetTime	Sets the time into a datetime value.
SetTimeZoneOffset	Provides a time zone portion of the datetime value.

Request Support DAC Forum Provide Feedback

5.18.1.7.2 Properties

Properties of the **TOraTimeStamp** class.

For a complete list of the **TOraTimeStamp** class members, see the TOraTimeStamp Members topic.

Public

Name	Description
AsDateTime	Used to get and set the timestamp value as TDateTime.
AsString	Used to get and set the timestamp value as string.
DescriptorType	Defines the type of the descriptor allocated to operate with the interval value.
Format	Used to get or set the date-time format model for operations with the TOraTimeStamp.AsString

	property.
IsNull	Used to get or set the null timestamp value flag.
OCIDateTime	Used to set the OCIDateTime descriptor handle.
OCIDateTimePtr	Used to provide the OCIDateTime type locator.
Precision	Used to learn or set the precision with which the fractional seconds are returned.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
TimeZone	Used to provide a time zone name portion of the datetime value.

See Also

- TOraTimeStamp Class
- TOraTimeStamp Class Members

Request Support DAC Forum Provide Feedback

5.18.1.7.2.1 AsDateTime Property

Used to get and set the timestamp value as TDateTime.

Class

TOraTimeStamp

Syntax

```
property AsDateTime: TDateTime;
```

Remarks

Use the AsDateTime property to get and set the timestamp value as TDateTime.

Reading the AsDateTime property when IsNull is True returns 0.

See Also

- IsNull

5.18.1.7.2.2 AsString Property

Used to get and set the timestamp value as string.

Class

TOraTimeStamp

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the timestamp value as string. Format of the string is specified by the Format property.

Reading the AsString property when IsNull is True returns empty string.

See Also

- Format
- IsNull

5.18.1.7.2.3 DescriptorType Property

Defines the type of the descriptor allocated to operate with the interval value.

Class

TOraTimeStamp

Syntax

```
property DescriptorType: cardinal;
```

Remarks

Use the DescriptorType property to define the type of the descriptor that is allocated to operate with the interval value. In most cases you don't need to set this property directly: it is adjusted automatically when working with fields and parameters.

Valid values for this property are:

OCI_DTYPE_TIMESTAMP,

OCI_DTYPE_TIMESTAMP_TZ,

OCI_DTYPE_TIMESTAMP_LTZ.

They are defined in the OraCall unit.

 Request Support DAC Forum Provide Feedback

5.18.1.7.2.4 Format Property

Used to get or set the date-time format model for operations with the AsString property.

Class

TOraTimeStamp

Syntax

```
property Format: string;
```

Remarks

Use the Format property to get or set the date-time format model for operations with the AsString property. For example, the date format model for string *'17:54:23'* is *'HH24:MM:SS'* . For complete description of the date-time models refer to the Oracle documentation.

See Also

- AsString

 Request Support DAC Forum Provide Feedback

Reserved.

5.18.1.7.2.5 IsNull Property

Used to get or set the null timestamp value flag.

Class

TOraTimeStamp

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to get or set the null timestamp value flag. When IsNull is true getting values via the GetDate, GetTime and SetTimeZoneOffset methods can raise an exception. The AsString property in this case returns empty string. The AsDateTime property returns 0.

See Also

- GetDate
- GetTime
- SetTimeZoneOffset
- AsString
- AsDateTime

5.18.1.7.2.6 OCIDateTime Property

Used to set the OCIDateTime descriptor handle.

Class

TOraTimeStamp

Syntax

```
property OCIDateTime: pOCIDateTime;
```

Remarks

Use the OCIDateTime property to get or set the OCIDateTime descriptor handle.

Request Support DAC Forum Provide Feedback

5.18.1.7.2.7 OCIDateTimePtr Property

Used to provide the OCIDateTime type locator.

Class

TOraTimeStamp

Syntax

```
property OCIDateTimePtr: ppOCIDateTime;
```

Remarks

Use the OCIDateTimePtr property to get the OCIDateTime type locator.

Request Support DAC Forum Provide Feedback

5.18.1.7.2.8 Precision Property

Used to learn or set the precision with which the fractional seconds are returned.

Class

TOraTimeStamp

Syntax

```
property Precision: byte;
```

Remarks

Use the Precision property to get or set the precision in which the fractional seconds are returned when reading the AsString property. The default value of the property is 6.

See Also

- AsString

 Request Support DAC Forum Provide Feedback

5.18.1.7.2.9 TimeZone Property

Used to provide a time zone name portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
property TimeZone: string;
```

Remarks

Use the TimeZone property to get a time zone name portion of the datetime value.

 Request Support DAC Forum Provide Feedback

5.18.1.7.3 Methods

Methods of the **TOraTimeStamp** class.

For a complete list of the **TOraTimeStamp** class members, see the TOraTimeStamp Members topic.

Protected

Name	Description
AllocDateTime	Allocates the OCIDateTime descriptor handle according to the TOraTimeStamp.DescriptorType property.

Public

Name	Description

AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
Compare	Compares the current TOraTimeStamp value with the Dest value.
Construct	Constructs datetime descriptor.
GetDate	Provides a date portion of the datetime value.
GetTime	Provides a time portion of the datetime value.
GetTimeZoneOffset	Provides a time zone portion of the datetime value.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SetDate	Provides a date portion of the datetime value.
SetTime	Sets the time into a datetime value.
SetTimeZoneOffset	Provides a time zone portion of the datetime value.

See Also

- TOraTimeStamp Class
- TOraTimeStamp Class Members

5.18.1.7.3.1 AllocDateTime Method

Allocates the OCIDateTime descriptor handle according to the DescriptorType property.

Class

TOraTimeStamp

Syntax

```
procedure AllocDateTime;
```

Remarks

Use the AllocDateTime method to allocate the OCIDateTime descriptor handle according to the DescriptorType property.

See Also

- DescriptorType

 Request Support DAC Forum Provide Feedback

5.18.1.7.3.2 Compare Method

Compares the current TOraTimeStamp value with the Dest value.

Class

TOraTimeStamp

Syntax

```
function Compare(Dest: TOraTimeStamp): integer;
```

Parameters

Dest

Holds the Dest value.

Return Value

negative value if the current timestamp is less than Dest, zero if timestamp is equal, and positive value if the current timestamp is greater than Dest.

Remarks

Call the Compare method to compare the current TOraTimeStamp value with the Dest value.

Returns negative value if current timestamp is less than Dest, zero if timestamp are equal and positive value if current timestamp is greater than Dest.

 Request Support DAC Forum Provide Feedback

5.18.1.7.3.3 Construct Method

Constructs datetime descriptor.

Class

TOraTimeStamp

Syntax

```
procedure Construct(Year: smallint; Month: byte; Day: byte; Hour:
byte; Min: byte; Sec: byte; FSec: cardinal; TimeZone: string);
```

Parameters

Year
Holds the year.

Month
Holds the month.

Day
Holds the day.

Hour
Holds the hour.

Min
Holds the minute.

Sec
Holds the second.

FSec
Holds the fraction second.

TimeZone
Holds the time zone name.

Remarks

Call the Construct method to construct datetime descriptor. When calling the Construct procedure only relevant fields based on the datetime type are used. For the type with time zone, the date and time fields are assumed to be in the local time of the specified time zone. If the time zone is not specified, then session default time zone is assumed.

See Also

- SetDate
- SetTime

- SetTimeZoneOffset

5.18.1.7.3.4 GetDate Method

Provides a date portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
procedure GetDate(var Year: smallint; var Month: byte; var Day: byte);
```

Parameters

Year
Holds the year.

Month
Holds the month.

Day
Holds the day.

Remarks

Call the GetDate method to get a date (year, month, day) portion of the datetime value.

See Also

- GetTime
- SetTimeZoneOffset

5.18.1.7.3.5 GetTime Method

Provides a time portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
procedure GetTime(var Hour: byte; var Min: byte; var Sec: byte;
var FSec: cardinal);
```

Parameters

Hour
Holds the hour.

Min
Holds the minute.

Sec
Holds the second.

FSec
Holds the fraction second.

Remarks

Call the GetTime method to get a time (hour, minute, second, fractional second) of the datetime value.

See Also

- GetDate
- SetTimeZoneOffset
- GetTime

5.18.1.7.3.6 GetTimeZoneOffset Method

Provides a time zone portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
procedure GetTimeZoneOffset(var TZHour: shortint; var TZMin:
```

```
shortint);
```

Parameters

TZHour

TZMin

Remarks

Call the GetTimeZoneOffset method to get a time zone (hour, minute) portion of the datetime value.

See Also

- GetDate
- GetTime

Request Support DAC Forum Provide Feedback

5.18.1.7.3.7 SetDate Method

Provides a date portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
procedure SetDate(Year: smallint; Month: byte; Day: byte);
```

Parameters

Year
Holds the year.

Month
Holds the month.

Day
Holds the day.

Remarks

Call the SetDate method to set a date (year, month, day) portion in a datetime value.

See Also

- Construct

5.18.1.7.3.8 SetTime Method

Sets the time into a datetime value.

Class

TOraTimeStamp

Syntax

```
procedure SetTime(Hour: byte; Min: byte; Sec: byte; FSec:
cardinal);
```

Parameters

Hour
Holds the hour.

Min
Holds the minute.

Sec
Holds the second.

FSec
Holds the fraction second.

Remarks

Call the SetTime method to set the time (hour, minute, second, fractional second) into a datetime value.

See Also

- Construct

Request Support DAC Forum Provide Feedback

5.18.1.7.3.9 SetTimeZoneOffset Method

Provides a time zone portion of the datetime value.

Class

TOraTimeStamp

Syntax

```
procedure SetTimeZoneOffset(TZHour: shortint; TZMin: shortint);
```

Parameters

TZHour
Holds the hour.

TZMin
Holds the minute.

Remarks

Call the SetTimeZoneOffset method to set a time zone (hour, minute) portion of the datetime value.

See Also

- Construct

Request Support DAC Forum Provide Feedback

5.18.2 Enumerations

Enumerations in the **OraClasses** unit.

Enumerations

Name	Description
TChangeNotifyEventType	Contains the event type.
TConnectMode	Specifies the system privileges used when a user connects to the server.
TMessageType	Defines the type of the next message found in the

	received named pipe local buffer.
TOptimizerMode	Specifies the optimizer mode for connection.

 Request Support DAC Forum Provide Feedback

5.18.2.1 TChangeNotifyEventType Enumeration

Contains the event type.

Unit

oraClasses

Syntax

```
TChangeNotifyEventType = (cneNone, cneStartup, cneShutdown,
cneShutdownAny, cneDropDB, cneDereg, cneObjChange, cneQueryChange);
```

Values

Value	Meaning
cneDereg	Registration has been removed.
cneDropDB	Database has been dropped.
cneNone	No further information about the continuous query notification.
cneObjChange	Object change notification.
cneQueryChange	Query change notification.
cneShutdown	Instance shutdown notification.
cneShutdownAny	Any instance shutdown when running RAC.
cneStartup	Instance startup notification.

 Request Support DAC Forum Provide Feedback

5.18.2.2 TConnectMode Enumeration

Specifies the system privileges used when a user connects to the server.

Unit

OraClasses

Syntax

```
TConnectMode = (cmNormal, cmSysOper, cmSysDBA, cmSysASM);
```

Values

Value	Meaning
cmNormal	Connect as an ordinary user. The default value.
cmSysASM	Connect with the SYSASM role.
cmSysDBA	Connect with the SYSDBA role.
cmSysOper	Connect with the SYSOPER role.

5.18.2.3 TMessageType Enumeration

Defines the type of the next message found in the received named pipe local buffer.

Unit

OraClasses

Syntax

```
TMessageType = (mtNone, mtNumber, mtString, mtDate);
```

Values

Value	Meaning
mtDate	Indicates that the messages found in the local buffer are of the Date type.
mtNone	Indicates that no more messages are found in the local buffer.
mtNumber	Indicates that the messages found in the local buffer are of the Number type.
mtString	Indicates that the messages found in the local buffer are of the String type.

5.18.2.4 TOptimizerMode Enumeration

Specifies the optimizer mode for connection.

Unit

OraClasses

Syntax

```
TOptimizerMode = (omDefault, omFirstRows1000, omFirstRows100,
omFirstRows10, omFirstRows1, omFirstRows, omAllRows, omChoose,
omRule);
```

Values

Value	Meaning
omAllRows	Explicitly chooses the cost-based approach to optimize a statement block with a goal of best throughput (that is minimum total resource consumption).
omChoose	Causes the optimizer to choose between the rule-based and cost-based approaches for a SQL statement. The optimizer selection is based on the presence of statistics for the tables accessed by the statement. If the data dictionary has statistics for at least one of these tables, then the optimizer uses the cost-based approach and optimizes with the goal of the best throughput. If the data dictionary does not have statistics for these tables, then it uses the rule-based approach.
omDefault	Session optimizer mode will not be changed.
omFirstRows	This mode is retained for backward compatibility and plan stability. It optimizes for the best plan to return the first single row.
omFirstRows1	Instruct Oracle to optimize a SQL statement for fast response. It instructs Oracle to choose the plan that returns the first row most efficiently. If you use the version server lower than Oracle 9.0, these values have the same effect as omFirstRows.
omFirstRows10	Instruct Oracle to optimize an SQL statement for fast response. It instructs Oracle to choose the plan that returns the first 10 rows most efficiently. If you use the version server lower than Oracle 9.0, these values have the same effect as omFirstRows.
omFirstRows100	Instruct Oracle to optimize an SQL statement for fast response. It instructs Oracle to choose the plan that returns the first 100 rows most efficiently. If you use the version server lower than Oracle 9.0, these values have the same effect as omFirstRows.
omFirstRows1000	Instruct Oracle to optimize an SQL statement for fast response. It

	instructs Oracle to choose the plan that returns the first 1000 rows most efficiently. If you use the version server lower than Oracle 9.0, these values have the same effect as omFirstRows.
omRule	Chooses rule-based optimization (RBO). Any other value causes the optimizer to choose cost-based optimization (CBO). The rule-based optimizer is the archaic optimizer mode from the earliest releases of Oracle Database.

Request Support DAC Forum Provide Feedback

5.18.3 Variables

Variables in the **OraClasses** unit.

Variables

Name	Description
FloatPrecision	Set this constant to define the type of NUMBER fields with Scale > 0.
IntegerPrecision	Set this constant to define the type of NUMBER fields with precision less than or equal to IntegerPrecision as dtInteger.
LargeIntPrecision	Set this constant to define the type of NUMBER fields with precision greater than the IntegerPresision and less than or equal to LargeIntPrecision as dtLargeInt.

Request Support DAC Forum Provide Feedback

5.18.3.1 FloatPrecision Variable

Set this constant to define the type of NUMBER fields with Scale > 0.

Unit

OraClasses

Syntax

```
FloatPrecision: integer = 15;
```

Remarks

Set this constant to define the type of NUMBER fields with Scale > 0. If its precision is less than or equal to FloatPrecision the field will be defined as TFloatField. Otherwise it will be defined as TOraNumberField (if TOraSessionOptions.EnableNumbers option is set to True).

According to these constants and TOraSessionOptions.EnableIntegers and TOraSessionOptions.EnableNumbers options, Oracle NUMBER type is mapped to ODAC field classes in the following way:

Conditions	Field class
Precision <= IntegerPrecision, Scale = 0, TOraSessionOptions.EnableIntegers = True	TIntegerField
IntegerPrecision < Precision <= LargeIntPrecision, Scale = 0, TOraSessionOptions.EnableIntegers = True	TLargeIntField
Precision > FloatPrecision, Scale > 0, TOraSessionOptions.EnableNumbers = True	TOraNumberField
In other cases	TFloatField

Example

```
FloatPrecision: integer = 15;
```


Request Support DAC Forum Provide Feedback

5.18.3.2 IntegerPrecision Variable

Set this constant to define the type of NUMBER fields with precision less than or equal to IntegerPrecision as dtInteger.

Unit

OraClasses

Syntax

```
IntegerPrecision: integer = 9;
```

 Request Support DAC Forum Provide Feedback

5.18.3.3 LargeIntPrecision Variable

Set this constant to define the type of NUMBER fields with precision greater than the IntegerPresision and less than or equal to LargeIntPrecision as dtLargeInt.

Unit

OraClasses

Syntax

```
LargeIntPrecision: integer = 0;
```

 Request Support DAC Forum Provide Feedback

5.19 OraConnectionPool

This unit contains the TOraConnectionPoolManager class for managing connection pool.

Enumerations

Name	Description
TOraPoolingType	Specifies the pool type.

 Request Support DAC Forum Provide Feedback

5.19.1 Enumerations

Enumerations in the **OraConnectionPool** unit.

Enumerations

Name	Description

TOraPoolingType	Specifies the pool type.

 Request Support DAC Forum Provide Feedback

5.19.1.1 TOraPoolingType Enumeration

Specifies the pool type.

Unit

OraConnectionPool

Syntax

```
TOraPoolingType = (optLocal, optOCI, optMTS);
```

Values

Value	Meaning
optLocal	Pool is created and controlled by ODAC.
optMTS	Pool is created and controlled by MTS.
optOCI	Pool is created and controlled by OCI.

 Request Support DAC Forum Provide Feedback

5.20 OraDataTypeMap

This unit contains the implementation of mapping between Oracle and Delphi data types.

Constants

Name	Description
oraAnyData	Used to map **ANYDATA** to Delphi data types.
oraBFile	Used to map **BFILE** to Delphi data types.
oraBinaryDouble	Used to map **BINARY_DOUBLE** to Delphi data types.

oraBinaryFloat	Used to map **BINARY_FLOAT** to Delphi data types.
oraBlob	Used to map **BLOB** to Delphi data types.
oraCFile	Used to map **CFILE** to Delphi data types.
oraChar	Used to map **CHAR** to Delphi data types.
oraClob	Used to map **CLOB** to Delphi data types.
oraCursor	Used to map **CURSOR** to Delphi data types.
oraDate	Used to map **DATE** to Delphi data types.
oraDoublePrecision	Used to map **DOUBLE PRECISION** to Delphi data types.
oraFloat	Used to map **FLOAT** to Delphi data types.
oraInteger	Used to map **INTEGER** to Delphi data types.
oraIntervalDS	Used to map **INTERVAL DAY TO SECOND** to Delphi data types.
oraIntervalYM	Used to map **INTERVAL YEAR TO MONTH** to Delphi data types.
oraLabel	Used to map **MLSLABEL** to Delphi data types.
oraLong	Used to map **LONG** to Delphi data types.
oraLongRaw	Used to map **LONG RAW** to Delphi data types.
oraNChar	Used to map **NCHAR** to Delphi data types.
oraNClob	Used to map **NCLOB** to Delphi data types.
oraNumber	Used to map **NUMBER** to Delphi data types.
oraNVarchar2	Used to map **NVARCHAR2** to Delphi data types.

oraObject	Used to map **OBJECT** to Delphi data types.
oraRaw	Used to map **RAW** to Delphi data types.
oraReference	Used to map **REF** to Delphi data types.
oraRowID	Used to map **ROWID** to Delphi data types.
oraTimeStamp	Used to map **TIMESTAMP** to Delphi data types.
oraTimeStampWithLocalTimeZone	Used to map **TIMESTAMP WITH LOCAL TIME ZONE** to Delphi data types.
oraTimeStampWithTimeZone	Used to map **TIMESTAMP WITH TIME ZONE** to Delphi data types.
oraUndefined	Used to map **UNDEFINED** to Delphi data types.
oraURowID	Used to map **UROWID** to Delphi data types.
oraVarchar2	Used to map **VARCHAR2** to Delphi data types.
oraXML	Used to map **XML** to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1 Constants

Constants in the **OraDataTypeMap** unit.

Constants

Name	Description
oraAnyData	Used to map **ANYDATA** to Delphi data types.
oraBFile	Used to map **BFILE** to Delphi data types.
oraBinaryDouble	Used to map **BINARY_DOUBLE** to Delphi data types.

oraBinaryFloat	Used to map **BINARY_FLOAT** to Delphi data types.
oraBlob	Used to map **BLOB** to Delphi data types.
oraCFile	Used to map **CFILE** to Delphi data types.
oraChar	Used to map **CHAR** to Delphi data types.
oraClob	Used to map **CLOB** to Delphi data types.
oraCursor	Used to map **CURSOR** to Delphi data types.
oraDate	Used to map **DATE** to Delphi data types.
oraDoublePrecision	Used to map **DOUBLE PRECISION** to Delphi data types.
oraFloat	Used to map **FLOAT** to Delphi data types.
oraInteger	Used to map **INTEGER** to Delphi data types.
oraIntervalDS	Used to map **INTERVAL DAY TO SECOND** to Delphi data types.
oraIntervalYM	Used to map **INTERVAL YEAR TO MONTH** to Delphi data types.
oraLabel	Used to map **MLSLABEL** to Delphi data types.
oraLong	Used to map **LONG** to Delphi data types.
oraLongRaw	Used to map **LONG RAW** to Delphi data types.
oraNChar	Used to map **NCHAR** to Delphi data types.
oraNClob	Used to map **NCLOB** to Delphi data types.
oraNumber	Used to map **NUMBER** to Delphi data types.
oraNVarchar2	Used to map **NVARCHAR2** to Delphi data types.

oraObject	Used to map **OBJECT** to Delphi data types.
oraRaw	Used to map **RAW** to Delphi data types.
oraReference	Used to map **REF** to Delphi data types.
oraRowID	Used to map **ROWID** to Delphi data types.
oraTimeStamp	Used to map **TIMESTAMP** to Delphi data types.
oraTimeStampWithLocalTimeZone	Used to map **TIMESTAMP WITH LOCAL TIME ZONE** to Delphi data types.
oraTimeStampWithTimeZone	Used to map **TIMESTAMP WITH TIME ZONE** to Delphi data types.
oraUndefined	Used to map **UNDEFINED** to Delphi data types.
oraURowID	Used to map **UROWID** to Delphi data types.
oraVarchar2	Used to map **VARCHAR2** to Delphi data types.
oraXML	Used to map **XML** to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.1 oraAnyData Constant

Used to map **ANYDATA** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraAnyData = oraBase + 31;
```

Remarks

Use the **oraAnyData** constant to map the Oracle **ANYDATA** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.2 oraBFile Constant

Used to map **BFILE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraBFile = oraBase + 20;
```

Remarks

Use the **oraBFile** constant to map the Oracle **BFILE** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.3 oraBinaryDouble Constant

Used to map **BINARY_DOUBLE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraBinaryDouble = oraBase + 9;
```

Remarks

Use the **oraBinaryDouble** constant to map the Oracle **BINARY_DOUBLE** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.4 oraBinaryFloat Constant

Used to map **BINARY_FLOAT** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraBinaryFloat = oraBase + 8;
```

Remarks

Use the **oraBinaryFloat** constant to map the Oracle **BINARY_FLOAT** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.5 oraBlob Constant

Used to map **BLOB** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraBlob = oraBase + 17;
```

Remarks

Use the **oraBlob** constant to map the Oracle **BLOB** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.6 oraCFile Constant

Used to map **CFILE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraCFile = oraBase + 21;
```

Remarks

Use the **oraCFile** constant to map the Oracle **CFILE** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.7 oraChar Constant

Used to map **CHAR** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraChar = oraBase + 1;
```

Remarks

Use the **oraChar** constant to map the Oracle **CHAR** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.8 oraClob Constant

Used to map **CLOB** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraClob = oraBase + 18;
```

Remarks

Use the **oraClob** constant to map the Oracle **CLOB** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.9 oraCursor Constant

Used to map **CURSOR** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraCursor = oraBase + 27;
```

Remarks

Use the **oraCursor** constant to map the Oracle **CURSOR** type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.10 oraDate Constant

Used to map **DATE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraDate = oraBase + 11;
```

Remarks

Use the **oraDate** constant to map the Oracle **DATE** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.11 oraDoublePrecision Constant

Used to map **DOUBLE PRECISION** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraDoublePrecision = oraBase + 10;
```

Remarks

Use the **oraDoublePrecision** constant to map the Oracle **DOUBLE PRECISION** data type to Delphi data types.

5.20.1.12 oraFloat Constant

Used to map **FLOAT** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraFloat = oraBase + 7;
```

Remarks

Use the **oraFloat** constant to map the Oracle **FLOAT** data type to Delphi data types.

5.20.1.13 oraInteger Constant

Used to map **INTEGER** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraInteger = oraBase + 6;
```

Remarks

Use the **oraInteger** constant to map the Oracle **INTEGER** data type to Delphi data types.

5.20.1.14 oraIntervalDS Constant

Used to map **INTERVAL DAY TO SECOND** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraIntervalDS = oraBase + 16;
```

Remarks

Use the **oraIntervalDS** constant to map the Oracle **INTERVAL DAY TO SECOND** data type to Delphi data types.

5.20.1.15 oraIntervalYM Constant

Used to map **INTERVAL YEAR TO MONTH** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraIntervalYM = oraBase + 15;
```

Remarks

Use the **oraIntervalYM** constant to map the Oracle **INTERVAL YEAR TO MONTH** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.16 oraLabel Constant

Used to map **MLSLABEL** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraLabel = oraBase + 32;
```

Remarks

Use the **oraLabel** constant to map the Oracle **MLSLABEL** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.17 oraLong Constant

Used to map **LONG** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraLong = oraBase + 22;
```

Remarks

Use the **oraLong** constant to map the Oracle **LONG** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.18 oraLongRaw Constant

Used to map **LONG RAW** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraLongRaw = oraBase + 24;
```

Remarks

Use the **oraLongRaw** constant to map the Oracle **LONG RAW** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.19 oraNChar Constant

Used to map **NCHAR** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraNChar = oraBase + 3;
```

Remarks

Use the **oraNChar** constant to map the Oracle **NCHAR** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.20 oraNClob Constant

Used to map **NCLOB** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraNClob = oraBase + 19;
```

Remarks

Use the **oraNClob** constant to map the Oracle **NCLOB** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.21 oraNumber Constant

Used to map **NUMBER** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraNumber = oraBase + 5;
```

Remarks

Use the **oraNumber** constant to map the Oracle **NUMBER** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.22 oraNVarchar2 Constant

Used to map **NVARCHAR2** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraNVarchar2 = oraBase + 4;
```

Remarks

Use the **oraNVarchar2** constant to map the Oracle **NVARCHAR2** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.23 oraObject Constant

Used to map **OBJECT** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraObject = oraBase + 28;
```

Remarks

Use the **oraObject** constant to map the Oracle **OBJECT** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.24 oraRaw Constant

Used to map **RAW** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraRaw = oraBase + 23;
```

Remarks

Use the **oraRaw** constant to map the Oracle **RAW** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.25 oraReference Constant

Used to map **REF** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraReference = oraBase + 29;
```

Remarks

Use the **oraReference** constant to map the Oracle **REF** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.26 oraRowID Constant

Used to map **ROWID** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraRowID = oraBase + 25;
```

Remarks

Use the **oraRowID** constant to map the Oracle **ROWID** data type to Delphi data types.

 Request Support DAC Forum Provide Feedback

5.20.1.27 oraTimeStamp Constant

Used to map **TIMESTAMP** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraTimeStamp = oraBase + 12;
```

Remarks

Use the **oraTimeStamp** constant to map the Oracle **TIMESTAMP** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.28 oraTimeStampWithLocalTimeZone Constant

Used to map **TIMESTAMP WITH LOCAL TIME ZONE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraTimeStampWithLocalTimeZone = oraBase + 14;
```

Remarks

Use the **oraTimeStampWithLocalTimeZone** constant to map the Oracle **TIMESTAMP WITH LOCAL TIME ZONE** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.29 oraTimeStampWithTimeZone Constant

Used to map **TIMESTAMP WITH TIME ZONE** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraTimeStampWithTimeZone = oraBase + 13;
```

Remarks

Use the **oraTimeStampWithTimeZone** constant to map the Oracle **TIMESTAMP WITH TIME ZONE** data type to Delphi data types.

5.20.1.30 oraUndefined Constant

Used to map **UNDEFINED** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraUndefined = oraBase + 33;
```

Remarks

Use the **oraUndefined** constant to map the Oracle **UNDEFINED** data type to Delphi data types.

5.20.1.31 oraURowID Constant

Used to map **UROWID** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraURowID = oraBase + 26;
```

Remarks

Use the **oraURowID** constant to map the Oracle **UROWID** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.32 oraVarchar2 Constant

Used to map **VARCHAR2** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraVarchar2 = oraBase + 2;
```

Remarks

Use the **oraVarchar2** constant to map the Oracle **VARCHAR2** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.20.1.33 oraXML Constant

Used to map **XML** to Delphi data types.

Unit

OraDataTypeMap

Syntax

```
oraXML = oraBase + 30;
```

Remarks

Use the **oraXML** constant to map the Oracle **XML** data type to Delphi data types.

Request Support DAC Forum Provide Feedback

5.21 OraErrHand

This unit contains the TOraErrorHandler component.

Classes

Name	Description
TOraErrorHandler	A component allowing translating of error messages.

Types

Name	Description
TOnOraErrorEvent	This type is used for the TOraErrorHandler.OnError event.

Request Support DAC Forum Provide Feedback

5.21.1 Classes

Classes in the **OraErrHand** unit.

Classes

Name	Description
TOraErrorHandler	A component allowing translating of error messages.

5.21.1.1 TOraErrorHandler Class

A component allowing translating of error messages.

For a list of all members of this type, see TOraErrorHandler members.

Unit

OraErrHand

Syntax

```
TOraErrorHandler = class(TComponent);
```

Remarks

TOraErrorHandler allows to translate error messages. Messages is stored in error table or can be got out of OnError event handler.

Structure of the error table must be as following:

```
ErrorCode   INTEGER
Constraint  VARCHAR2
Message     VARCHAR2
SenderName  VARCHAR2
ErrorClass  VARCHAR2
```

You can create and edit error table by ErrorHandler Editor in design time.

See Also

- EOraError

5.21.1.1.1 Members

TOraErrorHandler class overview.

Properties

Name	Description

Active	Activates searching messages in error table.
Debug	Used for an error message to contain additional information.
Session	Used to specify the session in which the dataset will be executed.
TableName	Used to define the error table name.

Methods

Name	Description
Close	Sets the Active property of a table to False.
Open	Sets the Active property to True.

Events

Name	Description
OnError	Occurs if an appropriate error can not be found in the error table or Active is False.

Request Support DAC Forum Provide Feedback

5.21.1.1.2 Properties

Properties of the **TOraErrorHandler** class.

For a complete list of the **TOraErrorHandler** class members, see the TOraErrorHandler Members topic.

Published

Name	Description
Active	Activates searching messages in error table.

Debug	Used for an error message to contain additional information.
Session	Used to specify the session in which the dataset will be executed.
TableName	Used to define the error table name.

See Also

- TOraErrorHandler Class
- TOraErrorHandler Class Members

 Request Support DAC Forum Provide Feedback

5.21.1.1.2.1 Active Property

Activates searching messages in error table.

Class

TOraErrorHandler

Syntax

```
property Active: boolean default False;
```

Remarks

When the Active property is True, ErrorHandler searches messages in error table.

See Also

- Open

 Request Support DAC Forum Provide Feedback

5.21.1.1.2.2 Debug Property

Used for an error message to contain additional information.

Class

TOraErrorHandler

Syntax

```
property Debug: boolean default False;
```

Remarks

When the Debug property is True, error message contains additional information: sender name, constraint name and error code.

Note: To use this property you should explicitly include OdacVcl (OdacClx under Linux) unit to your project.

If TOraSQLMonitor is used in the project and the TOraSQLMonitor.Active property is set to False, the debug window is not displayed.

Request Support DAC Forum Provide Feedback

5.21.1.1.2.3 Session Property

Used to specify the session in which the dataset will be executed.

Class

TOraErrorHandler

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session in which the dataset will be executed. If Session is not connected, the Open method calls Session.Connect.

See Also

- TOraSession

Request Support DAC Forum Provide Feedback

5.21.1.1.2.4 TableName Property

Used to define the error table name.

Class

TOraErrorHandler

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to determine the error table name. The error table must exist.

Request Support DAC Forum Provide Feedback

5.21.1.1.3 Methods

Methods of the **TOraErrorHandler** class.

For a complete list of the **TOraErrorHandler** class members, see the TOraErrorHandler Members topic.

Public

Name	Description
Close	Sets the Active property of a table to False.
Open	Sets the Active property to True.

See Also

- TOraErrorHandler Class
- TOraErrorHandler Class Members

5.21.1.1.3.1 Close Method

Sets the Active property of a table to False.

Class

TOraErrorHandler

Syntax

```
procedure Close;
```

Remarks

Call the Close method to set the Active property of a table to False. When Active is False, the table is closed; the table cannot read data from or write data to the database.

See Also

- Open

5.21.1.1.3.2 Open Method

Sets the Active property to True.

Class

TOraErrorHandler

Syntax

```
procedure Open;
```

Remarks

Call the Open method to set the Active property for the table to True. When Active is True, data can be read from and written to the database.

See Also

- Close
- Active

Request Support DAC Forum Provide Feedback

5.21.1.1.4 Events

Events of the **TOraErrorHandler** class.

For a complete list of the **TOraErrorHandler** class members, see the TOraErrorHandler Members topic.

Published

Name	Description
OnError	Occurs if an appropriate error can not be found in the error table or Active is False.

See Also

- TOraErrorHandler Class
- TOraErrorHandler Class Members

Request Support DAC Forum Provide Feedback

5.21.1.1.4.1 OnError Event

Occurs if an appropriate error can not be found in the error table or Active is False.

Class

TOraErrorHandler

Syntax

```
property OnError: TOnOraErrorEvent;
```

Remarks

Occurs when ErrorHandler can not find an appropriate error in the error table or Active is False.

See Also

- TCustomDAConnection.OnError

 Request Support DAC Forum Provide Feedback

5.21.2 Types

Types in the **OraErrHand** unit.

Types

Name	Description
TOnOraErrorEvent	This type is used for the TOraErrorHandler.OnError event.

 Request Support DAC Forum Provide Feedback

5.21.2.1 TOnOraErrorEvent Procedure Reference

This type is used for the TOraErrorHandler.OnError event.

Unit

OraErrHand

Syntax

```
TOnOraErrorEvent = procedure (Sender: TObject; E: Exception;
ErrorCode: integer; const ConstraintName: string; var Msg:
string) of object;
```

Parameters

Sender

An object that raised the event.

E
Holds the reference to an exception object.

ErrorCode
The code of an error.

ConstraintName
Holds the name of the constraint that raised the error.

Msg
Holds the error message (can be set individually).

5.22 OraError

This unit contains the EOraError exception class.

Classes

Name	Description
EOraError	Raised when a component detects Oracle error.

5.22.1 Classes

Classes in the **OraError** unit.

Classes

Name	Description
EOraError	Raised when a component detects Oracle error.

5.22.1.1 EOraError Class

Raised when a component detects Oracle error.

For a list of all members of this type, see EOraError members.

Unit

OraError

Syntax

```
EOraError = class(EDAError);
```

Remarks

EOraError is raised when a component detects Oracle error. Use EOraError in an exception handling block.

Inheritance Hierarchy

EDAError

EOraError

See Also

- TOraErrorHandler

5.22.1.1.1 Members

EOraError class overview.

Properties

Name	Description
Component (inherited from **EDAError**)	Contains the component that caused the error.
ErrorCode (inherited from **EDAError**)	Determines the error code returned by the server.
Sender	Holds reference to the sender if exception is raised by the TComponent instance.

5.22.1.1.2 Properties

Properties of the **EOraError** class.

For a complete list of the **EOraError** class members, see the EOraError Members topic.

Public

Name	Description
Component (inherited from **EDAError**)	Contains the component that caused the error.
ErrorCode (inherited from **EDAError**)	Determines the error code returned by the server.
Sender	Holds reference to the sender if exception is raised by the TComponent instance.

See Also

- EOraError Class
- EOraError Class Members

Request Support DAC Forum Provide Feedback

5.22.1.1.2.1 Sender Property

Holds reference to the sender if exception is raised by the TComponent instance.

Class

`EOraError`

Syntax

```
property Sender: TComponent;
```

Remarks

The Sender property holds reference to the sender if exception is raised by the TComponent

instance.

5.23 OraLoader

This unit contains implementation of the TOraLoader component.

Classes

Name	Description
TDPColumn	A base class holding a collection of TDPColumn objects.
TOraLoader	TOraLoader allows to load external data into the server database.

Types

Name	Description
TDPErrorEvent	This type is used for the TOraLoader.OnError event.
TDPGetColumnDataEvent	This type is used for the TOraLoader.OnGetColumnData event.
TDPPutDataEvent	This type is used for the TOraLoader.OnPutData event.

Enumerations

Name	Description
TDPErrorAction	Specifies the action for TOraLoader to take to process errors that occur during loading.
TLoadMode	Specifies the access mode to use for a TOraLoader object when a database table is being modified.

 Request Support DAC Forum Provide Feedback

5.23.1 Classes

Classes in the **OraLoader** unit.

Classes

Name	Description
TDPColumn	A base class holding a collection of TDPColumn objects.
TOraLoader	TOraLoader allows to load external data into the server database.

 Request Support DAC Forum Provide Feedback

5.23.1.1 TDPColumn Class

A base class holding a collection of TDPColumn objects.

For a list of all members of this type, see TDPColumn members.

Unit

OraLoader

Syntax

```
TDPColumn = class(TDAColumn);
```

Remarks

Each TOraLoader uses TDPColumns to maintain a collection of TDPColumn objects. TDPColumn object represents the attributes for column loading. Every TDPColumn object corresponds to one of the table fields with the same name as its Name property.

To create columns at design time use column editor of TOraLoader component.

Inheritance Hierarchy

TDAColumn

TDPColumn

See Also

- TOraLoader
- TQueueAgents

 Request Support DAC Forum Provide Feedback

5.23.1.1.1 Members

TDPColumn class overview.

Properties

Name	Description
DateFormat	Used to specify a conversion mask for a column.
FieldType (inherited from **TDAColumn**)	Used to specify the types of values that will be loaded.
Name (inherited from **TDAColumn**)	Used to specify the field name of loading table.
Precision	Used to set the precision of number columns.
Scale	Used to set the precision of number columns.
Size	Used to set the maximum size in bytes of data for a column.

 Request Support DAC Forum Provide Feedback

5.23.1.1.2 Properties

Properties of the **TDPColumn** class.

For a complete list of the **TDPColumn** class members, see the TDPColumn Members topic.

Published

Name	Description
DateFormat	Used to specify a conversion mask for a column.
FieldType (inherited from **TDAColumn**)	Used to specify the types of values that will be loaded.
Name (inherited from **TDAColumn**)	Used to specify the field name of loading table.
Precision	Used to set the precision of number columns.
Scale	Used to set the precision of number columns.
Size	Used to set the maximum size in bytes of data for a column.

See Also

- TDPColumn Class
- TDPColumn Class Members

Request Support DAC Forum Provide Feedback

5.23.1.1.2.1 DateFormat Property

Used to specify a conversion mask for a column.

Class

TDPColumn

Syntax

```
property DateFormat: string;
```

Remarks

Set DateFormat to specify a conversion mask for a column. TOraLoader uses DateFormat to convert string representation of date to its internal representation. If not set, the date format

defaults to the date conversion mask set in the direct path context.

See Also

- TDAColumn.Name

 Request Support DAC Forum Provide Feedback

5.23.1.1.2.2 Precision Property

Used to set the precision of number columns.

Class

TDPColumn

Syntax

```
property Precision: integer default 0;
```

Remarks

Use Precision property to set the precision of number columns.

See Also

- TDAColumn.FieldType

 Request Support DAC Forum Provide Feedback

5.23.1.1.2.3 Scale Property

Used to set the precision of number columns.

Class

TDPColumn

Syntax

```
property Scale: integer default 0;
```

Remarks

Use Scale property to set the scale of number columns.

See Also

- TDAColumn.FieldType

 Request Support DAC Forum Provide Feedback

5.23.1.1.2.4 Size Property

Used to set the maximum size in bytes of data for a column.

Class

TDPColumn

Syntax

```
property Size: integer default 0;
```

Remarks

Use Size property to set the maximum size in bytes of data for a column. Size is used only for string columns.

See Also

- TDAColumn.FieldType

 Request Support DAC Forum Provide Feedback

5.23.1.2 TOraLoader Class

TOraLoader allows to load external data into the server database.

For a list of all members of this type, see TOraLoader members.

Unit

OraLoader

Syntax

```
TOraLoader = class(TDALoader);
```

Remarks

TOraLoader serves for fast loading of data to the server. It uses direct path load interface to speed up loading. To specify the name of the loading table set TDALoader.TableName property. Use TDALoader.Columns property to access individual columns. Write TDALoader.OnGetColumnData or TDALoader.OnPutData event handlers to read external data and pass it to the database. Call TDALoader.Load method to start loading data.

Limitations and Restrictions

TOraLoader has the following limitations similar to those of SQL*Loader:

- triggers are not supported
- check constraints are not supported
- referential integrity constraints are not supported
- clustered tables are not supported
- loading of remote objects is not supported
- user-defined types are not supported
- LOBs must be specified after all scalar columns
- LONGs must be specified last
- You cannot use TOraLoader in a threaded OCI environment in direct mode with Oracle client 8.17 or lower.
- ODAC sets T:Devart.Odac.Units.OraCall := False when you use OraLoader unit in your application. You can set it to True for Oracle client 9.2 or higher.

Inheritance Hierarchy

TDALoader

  TOraLoader

See Also

- TOraLoader Component
- TDALoader

- TQueueAgents
- TOraSession

5.23.1.2.1 Members

TOraLoader class overview.

Properties

Name	Description
Columns (inherited from **TDALoader**)	Used to add a TDAColumn object for each field that will be loaded.
Connection (inherited from **TDALoader**)	See the TOraLoader.Session property.
LoadMode	Used to specify which access mode to use for a TOraLoader object when a database table is being modified.
Session	Used to specify the session in which TOraLoader will be executed.
TableName (inherited from **TDALoader**)	Used to specify the name of the table to which data will be loaded.

Methods

Name	Description
CreateColumns (inherited from **TDALoader**)	Creates TDAColumn objects for all fields of the table with the same name as TDALoader.TableName.
Load (inherited from **TDALoader**)	Starts loading data.
LoadFromDataSet (inherited from **TDALoader**)	Loads data from the specified dataset.

PutColumnData (inherited from **TDALoader**)	Overloaded. Puts the value of individual columns.

Events

Name	Description
OnError	Occurs when processing errors that are raised during loading is needed.
OnGetColumnData	Occurs when it is needed to put column values.
OnProgress (inherited from **TDALoader**)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.
OnPutData	Occurs when putting loading data by rows is needed.

Request Support DAC Forum Provide Feedback

5.23.1.2.2 Properties

Properties of the **TOraLoader** class.

For a complete list of the **TOraLoader** class members, see the TOraLoader Members topic.

Public

Name	Description
Columns (inherited from **TDALoader**)	Used to add a TDAColumn object for each field that will be loaded.
Connection (inherited from **TDALoader**)	See the TOraLoader.Session property.
LoadMode	Used to specify which access mode to use for a TOraLoader object when a database table is being modified.
TableName (inherited from **TDALoader**)	Used to specify the name of the table to which data will

	be loaded.

Published

Name	Description
Session	Used to specify the session in which TOraLoader will be executed.

See Also

- TOraLoader Class
- TOraLoader Class Members

Request Support DAC Forum Provide Feedback

5.23.1.2.2.1 LoadMode Property

Used to specify which access mode to use for a TOraLoader object when a database table is being modified.

Class

TOraLoader

Syntax

```
property LoadMode: TLoadMode;
```

Remarks

Use the LoadMode property to specify which access mode to use for a TOraLoader object when a database table is being modified.

Set this property to lmDirect to make all modifications pass through internal data buffers or set it to lmDML to construct relevant DML statement which applies updates to the database table.

See Also

- TOraLoader Component

5.23.1.2.2.2 Session Property

Used to specify the session in which TOraLoader will be executed.

Class

TOraLoader

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session in which TOraLoader will be executed. If Session is not connected, Load method calls Session.Connect.

See Also

- TOraSession

5.23.1.2.3 Events

Events of the **TOraLoader** class.

For a complete list of the **TOraLoader** class members, see the TOraLoader Members topic.

Public

Name	Description
OnProgress (inherited from **TDALoader**)	Occurs if handling data loading progress of the TDALoader.LoadFromDataSet method is needed.

Published

Name	Description
OnError	Occurs when processing errors that are raised during loading is needed.
OnGetColumnData	Occurs when it is needed to put column values.
OnPutData	Occurs when putting loading data by rows is needed.

See Also

- TOraLoader Class
- TOraLoader Class Members

Request Support DAC Forum Provide Feedback

5.23.1.2.3.1 OnError Event

Occurs when processing errors that are raised during loading is needed.

Class

TOraLoader

Syntax

```
property OnError: TDPErrorEvent;
```

Remarks

Write the OnError event handler to process errors that occur during loading. Handler is used only if LoadMode = lmDirect, i.e. when using Oracle Direct Path interface. E parameter is an exception that was raised. Col and Row parameters are appropriate column and row values for data that loader failed to write to a table.

See Also

- LoadMode

Request Support DAC Forum Provide Feedback

5.23.1.2.3.2 OnGetColumnData Event

Occurs when it is needed to put column values.

Class

TOraLoader

Syntax

```
property OnGetColumnData: TDPGetColumnDataEvent;
```

Remarks

Write OnGetColumnData event handler to put column values. TOraLoader calls OnGetColumnData event handler for each column in the loop. Column points to TDAColumn object that corresponds to the current loading column. Use its Name or Index property to identify what column is loading. The Row parameter indicates the current loading record. TDALoader increments Row parameter when all the columns of the current record are loaded. The first row is 1. Set EOF to True to stop data loading. Fill Value parameter by column values. To start loading call the TDALoader.Load method.

Another way to load data is using OnPutData event.

See Also

- TDALoader.OnPutData
- TDALoader.Load
- OnPutData

5.23.1.2.3.3 OnPutData Event

Occurs when putting loading data by rows is needed.

Class

TOraLoader

Syntax

```
property OnPutData: TDPPutDataEvent;
```

Remarks

Note that rows should be loaded from the first in ascending order. To start loading, call TDALoader.Load method.

To start loading, call the TDALoader.Load method.

See Also

- TDALoader.PutColumnData
- TDALoader.Load
- TDALoader.OnGetColumnData
- OnGetColumnData

 Request Support DAC Forum Provide Feedback

5.23.2 Types

Types in the **OraLoader** unit.

Types

Name	Description
TDPErrorEvent	This type is used for the TOraLoader.OnError event.
TDPGetColumnDataEvent	This type is used for the TOraLoader.OnGetColumnData event.
TDPPutDataEvent	This type is used for the TOraLoader.OnPutData event.

 Request Support DAC Forum Provide Feedback

5.23.2.1 TDPErrorEvent Procedure Reference

This type is used for the TOraLoader.OnError event.

Unit

OraLoader

Syntax

```
TDPErrorEvent = procedure (Sender: TOraLoader; E: Exception; Col:
integer; Row: integer; var Action: TDPErrorAction) of object;
```

Parameters

Sender
An object that raised the event.

E
The exception that was raised.

Col
The column value for data that loader failed to write to a table.

Row
The row value for data that loader failed to write to a table.

Action
The action to take when an exception is raised.

Request Support DAC Forum Provide Feedback

5.23.2.2 TDPGetColumnDataEvent Procedure Reference

This type is used for the TOraLoader.OnGetColumnData event.

Unit

OraLoader

Syntax

```
TDPGetColumnDataEvent = procedure (Sender: TObject; Column:
TDPColumn; Row: integer; var Value: variant; var IsEOF: boolean)
of object;
```

Parameters

Sender

An object that raised the event.

Column

Points to TDAColumn object that corresponds to the current loading column.

Row

Indicates the current loading record.

Value

The column values.

IsEOF

True, if data loading needs to be stopped.

 Request Support DAC Forum Provide Feedback

5.23.2.3 TDPPutDataEvent Procedure Reference

This type is used for the TOraLoader.OnPutData event.

Unit

OraLoader

Syntax

```
TDPPutDataEvent = procedure (Sender: TOraLoader) of object;
```

Parameters

Sender

An object that raised the event.

 Request Support DAC Forum Provide Feedback

5.23.3 Enumerations

Enumerations in the **OraLoader** unit.

Enumerations

Name	Description
TDPErrorAction	Specifies the action for TOraLoader to take to process errors that occur during loading.

TLoadMode	Specifies the access mode to use for a TOraLoader object when a database table is being modified.

 Request Support DAC Forum Provide Feedback

5.23.3.1 TDPErrorAction Enumeration

Specifies the action for TOraLoader to take to process errors that occur during loading.

Unit

OraLoader

Syntax

```
TDPErrorAction = (dpAbort, dpFail, dpIgnore);
```

Values

Value	Meaning
dpAbort	TOraLoader silently interrupts the current operation.
dpFail	TOraLoader reraises the exception passed as E parameter.
dpIgnore	TOraLoader skips the current row and continues execution.

 Request Support DAC Forum Provide Feedback

5.23.3.2 TLoadMode Enumeration

Specifies the access mode to use for a TOraLoader object when a database table is being modified.

Unit

OraLoader

Syntax

```
TLoadMode = (lmDirect, lmDML);
```

Values

Value	Meaning
lmDirect	All modifications are passed through internal data buffers.
lmDML	Constructs relevant DML statement which applies updates to the database table.

Remarks

Use the LoadMode property to specify which access mode to use for a TOraLoader object when a database table is being modified.

Set this property to lmDirect to make all modifications pass through internal data buffers or set it to lmDML to construct relevant DML statement which applies updates to the database table.

Request Support DAC Forum Provide Feedback

5.24 OraNet

This unit implements the Direct Mode in ODAC.

Enumerations

Name	Description
TSecurityLevel	Used to turn on the Oracle Advanced Security encryption and integrity.

Variables

Name	Description
DataIntegrityLevel	Use this variable to prevent unauthorized data modification when it is passed over the network.
EncryptionLevel	Use this variable to prevent unauthorized data viewing when it is passed over the network.

PacketSize	Use this constant to specify the size of transferred packets.

 Request Support DAC Forum Provide Feedback

5.24.1 Enumerations

Enumerations in the **OraNet** unit.

Enumerations

Name	Description
TSecurityLevel	Used to turn on the Oracle Advanced Security encryption and integrity.

 Request Support DAC Forum Provide Feedback

5.24.1.1 TSecurityLevel Enumeration

Used to turn on the Oracle Advanced Security encryption and integrity.

Unit

OraNet

Syntax

```
TSecurityLevel = (slAccepted = 0, slRejected = 1, slRequested = 2,
slRequired = 3);
```

Values

Value	Meaning
slAccepted	Select this value to enable the security service if required or requested by the other side.
slRejected	Select this value if you do not elect to enable the security service, even if required by the other side.
slRequested	Select this value to enable the security service if the other side permits it.

sIRequired	Select this value to enable the security service or preclude the connection.

Request Support DAC Forum Provide Feedback

5.24.2 Variables

Variables in the **OraNet** unit.

Variables

Name	Description
DataIntegrityLevel	Use this variable to prevent unauthorized data modification when it is passed over the network.
EncryptionLevel	Use this variable to prevent unauthorized data viewing when it is passed over the network.
PacketSize	Use this constant to specify the size of transferred packets.

Request Support DAC Forum Provide Feedback

5.24.2.1 DataIntegrityLevel Variable

Use this variable to prevent unauthorized data modification when it is passed over the network.

Unit

OraNet

Syntax

```
DataIntegrityLevel: TSecurityLevel;
```

Remarks

Use the DataIntegrityLevel variable to prevent unauthorized data modification when it is passed over the network. To select the data integrity level, specify one of the possible values of the TSecurityLevel variable. The default value is Accepted.

 Request Support DAC Forum Provide Feedback

5.24.2.2 EncryptionLevel Variable

Use this variable to prevent unauthorized data viewing when it is passed over the network.

Unit

OraNet

Syntax

```
EncryptionLevel: TSecurityLevel;
```

Remarks

Use the EncryptionLevel variable to prevent unauthorized data viewing when it is passed over the network. To select the data encryption level, specify one of the existing values of the TSecurityLevel variable. The default value is Accepted.

 Request Support DAC Forum Provide Feedback

5.24.2.3 PacketSize Variable

Use this constant to specify the size of transferred packets.

Unit

OraNet

Syntax

```
PacketSize: Integer;
```

Remarks

Use this constant to specify the size of transferred packets. If you set this variable to an

optimal value (that depends on the network you are working in), this can significantly increase performance for VPN, Wireless and other networks.

Allowed values are within the range from 512 to 65535 bytes. The default value is 8192 bytes.

 Request Support DAC Forum Provide Feedback

5.25 OraObjects

This unit contains classes for Oracle OBJECT, ARRAY, TABLE and XMLTYPE data types.

Classes

Name	Description
TOraArray	A class representing the value of the Oracle array data type.
TOraNestTable	A class representing a value of the Oracle nested table data type.
TOraObject	A class representing the Oracle object data type value.
TOraRef	A class representing the Oracle reference data type value.
TOraType	A class holding information about Oracle type required for TOraObject objects.
TOraXML	A class representing a value of the Oracle SYS.XMLTYPE type.

 Request Support DAC Forum Provide Feedback

5.25.1 Classes

Classes in the **OraObjects** unit.

Classes

Name	Description
TOraArray	A class representing the value of the Oracle array data type.
TOraNestTable	A class representing a value of the Oracle nested table data type.
TOraObject	A class representing the Oracle object data type value.
TOraRef	A class representing the Oracle reference data type value.
TOraType	A class holding information about Oracle type required for TOraObject objects.
TOraXML	A class representing a value of the Oracle SYS.XMLTYPE type.

Request Support DAC Forum Provide Feedback

5.25.1.1 TOraArray Class

A class representing the value of the Oracle array data type.

For a list of all members of this type, see TOraArray members.

Unit

OraObjects

Syntax

```
TOraArray = class(TOraObject);
```

Remarks

TOraArray represents the value of the Oracle array data type. TOraArray is inherited from TOraObject. You can get a TOraArray object by the TOraDataSet.GetArray method after fetching rows containing array field.

Inheritance Hierarchy

TSharedObject
TDBObject
TOraObject
TOraArray

See Also

- TOraObject
- TOraDataSet.GetArray
- TOraParam.AsArray

Request Support DAC Forum Provide Feedback

5.25.1.1.1 Members

TOraArray class overview.

Properties

Name	Description
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or

	to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ItemAsAnsiString	Used to read the value of the array element into an AnsiString or to assign an AnsiString to the array element.
ItemAsDateTime	Used to read the value of the array element into a TDateTime variable or to assign a TDateTime variable to the array element.
ItemAsFloat	Used to read the value of the array element into a Double or to assign a Double to the

	array element.
ItemAsInteger	Used to read the value of the array element into an Integer or to assign an Integer to the array element.
ItemAsInterval	Used to read the value of the array element into a TOraIntervalI variable or to assign a TOraInterval variable to the array element.
ItemAsLargeInt	Used to read the value of the array element into an Int64 or to assign an Int64 to the array element.
ItemAsLob	Used to read the value of the array element into a TOraLob variable or to assign a TOraLob variable to the array element.
ItemAsObject	Used to read the value of the array element into a TOraObject variable or to assign a TOraObject variable to the array element.
ItemAsOCIString	Used to read the value of the array element into a generic OCIString pointer or to assign the OCIString pointer to the array element.
ItemAsRef	Used to read the value of the array element into a TOraRef variable or to assign a TOraRef variable to the array element.
ItemAsString	Used to read the value of the array element into a String or to assign a String to the array element.
ItemAsTimeStamp	Used to read the value of the array element into a TOraTimeStampI variable or to assign a TOraTimeStamp variable to the array

	element.
ItemAsWideString	Used to read the value of the array element into a WideString or to assign a WideString to the array element.
ItemExists	Indicates whether an element with the specified index exists in the array.
ItemIsNull	Indicates whether the array element contains a value.
ItemSubType	Indicates the subtype of an array element.
ItemType	Indicates the type of an array element.
MaxSize	Defines the maximum number of elements that an array object may hold.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size	Holds the size of an array.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
AppendItem	Appends an element to the end of an array.
Assign	Copies properties and attributes of one object to another one.
Clear	Removes all elements from an array.

CreateObject (inherited from **TOraObject**)	Creates an object.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
InsertItem	Inserts an element at a specific index.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.

Request Support DAC Forum Provide Feedback

5.25.1.1.2 Properties

Properties of the **TOraArray** class.

For a complete list of the **TOraArray** class members, see the TOraArray Members topic.

Public

Name	Description
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.

AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.

Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ItemAsAnsiString	Used to read the value of the array element into an AnsiString or to assign an AnsiString to the array element.
ItemAsDateTime	Used to read the value of the array element into a TDateTime variable or to assign a TDateTime variable to the array element.
ItemAsFloat	Used to read the value of the array element into a Double or to assign a Double to the array element.
ItemAsInteger	Used to read the value of the array element into an Integer or to assign an Integer to the array element.
ItemAsInterval	Used to read the value of the array element into a TOraIntervalI variable or to assign a TOraInterval variable to the array element.
ItemAsLargeInt	Used to read the value of the array element into an Int64 or to assign an Int64 to the array element.
ItemAsLob	Used to read the value of the array element into a TOraLob variable or to assign a TOraLob variable to the array element.
ItemAsObject	Used to read the value of the array element into a TOraObject variable or to assign a TOraObject variable to the array element.

ItemAsOCIString	Used to read the value of the array element into a generic OCIString pointer or to assign the OCIString pointer to the array element.
ItemAsRef	Used to read the value of the array element into a TOraRef variable or to assign a TOraRef variable to the array element.
ItemAsString	Used to read the value of the array element into a String or to assign a String to the array element.
ItemAsTimeStamp	Used to read the value of the array element into a TOraTimeStampl variable or to assign a TOraTimeStamp variable to the array element.
ItemAsWideString	Used to read the value of the array element into a WideString or to assign a WideString to the array element.
ItemExists	Indicates whether an element with the specified index exists in the array.
ItemIsNull	Indicates whether the array element contains a value.
ItemSubType	Indicates the subtype of an array element.
ItemType	Indicates the type of an array element.
MaxSize	Defines the maximum number of elements that an array object may hold.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Size	Holds the size of an array.

See Also

- TOraArray Class
- TOraArray Class Members

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.1 ItemAsAnsiString Property(Indexer)

Used to read the value of the array element into an AnsiString or to assign an AnsiString to the array element.

Class

TOraArray

Syntax

```
property ItemAsAnsiString[Index: integer]: AnsiString;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsAnsiString property to read the value of the array element into an AnsiString or to assign an AnsiString to the array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.2 ItemAsDateTime Property(Indexer)

Used to read the value of the array element into a TDateTime variable or to assign a TDateTime variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsDateTime[Index: integer]: TDateTime;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsDateTime property to read the value of the array element into a TDateTime varible or to assign a TDateTime variable to the array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.3 ItemAsFloat Property (Indexer)

Used to read the value of the array element into a Double or to assign a Double to the array element.

Class

TOraArray

Syntax

```
property ItemAsFloat[Index: integer]: double;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsFloat property to read the value of the array element into a Double or to assign a Double to the array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.4 ItemAsInteger Property(Indexer)

Used to read the value of the array element into an Integer or to assign an Integer to the array element.

Class

TOraArray

Syntax

```
property ItemAsInteger[Index: integer]: integer;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsInteger property to read the value of the array element into an Integer or to assign an Integer to the array element.

Request Support DAC Forum Provide Feedback

5.25.1.1.2.5 ItemAsInterval Property(Indexer)

Used to read the value of the array element into a TOraIntervalI variable or to assign a TOraInterval variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsInterval[Index: integer]: TOraInterval;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsInterval property to read the value of the array element into a TOraInterval

variable or to assign a TOraInterval variable to the array element.

See Also

- TOraInterval

5.25.1.1.2.6 ItemAsLargeInt Property(Indexer)

Used to read the value of the array element into an Int64 or to assign an Int64 to the array element.

Class

TOraArray

Syntax

```
property ItemAsLargeInt[Index: integer]: int64;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsLargeInt property to read the value of the array element into an Int64 or to assign an Int64 to the array element.

5.25.1.1.2.7 ItemAsLob Property(Indexer)

Used to read the value of the array element into a TOraLob variable or to assign a TOraLob variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsLob[Index: integer]: TOraLob;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsLob property to read the value of the array element into a TOraLob variable or to assign a TOraLob variable to the array element.

See Also

- TOraLob

5.25.1.1.2.8 ItemAsObject Property(Indexer)

Used to read the value of the array element into a TOraObject variable or to assign a TOraObject variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsObject[Index: integer]: TOraObject;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsObject property to read the value of the array element into a TOraObject variable or to assign a TOraObject variable to the array element.

See Also

- TOraObject

5.25.1.1.2.9 ItemAsOCIString Property(Indexer)

Used to read the value of the array element into a generic OCIString pointer or to assign the OCIString pointer to the array element.

Class

TOraArray

Syntax

```
property ItemAsOCIString[Index: integer]: pOCIString;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsOCIString property to read the value of the array element into a generic OCIString pointer or to assign the OCIString pointer to the array element.

Request Support DAC Forum Provide Feedback

5.25.1.1.2.10 ItemAsRef Property(Indexer)

Used to read the value of the array element into a TOraRef variable or to assign a TOraRef variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsRef[Index: integer]: TOraRef;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsRef property to read the value of the array element into a TOraRef variable or to assign a TOraRef variable to the array element.

See Also

- TOraRef

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.11 ItemAsString Property(Indexer)

Used to read the value of the array element into a String or to assign a String to the array element.

Class

TOraArray

Syntax

```
property ItemAsString[Index: integer]: string;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsString property to read the value of the array element into a String or to assign a String to the array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.12 ItemAsTimeStamp Property(Indexer)

Used to read the value of the array element into a TOraTimeStampl variable or to assign a TOraTimeStamp variable to the array element.

Class

TOraArray

Syntax

```
property ItemAsTimeStamp[Index: integer]: TOraTimeStamp;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsTimeStamp property to read the value of the array element into a TOraTimeStamp variable or to assign a TOraTimeStamp variable to the array element.

See Also

- TOraTimeStamp

Request Support DAC Forum Provide Feedback

5.25.1.1.2.13 ItemAsWideString Property(Indexer)

Used to read the value of the array element into a WideString or to assign a WideString to the array element.

Class

TOraArray

Syntax

```
property ItemAsWideString[Index: integer]: string;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemAsWideString property to read the value of the array element into a WideString or to assign a WideString to the array element.

Request Support DAC Forum Provide Feedback

5.25.1.1.2.14 ItemExists Property(Indexer)

Indicates whether an element with the specified index exists in the array.

Class

TOraArray

Syntax

```
property ItemExists[Index: integer]: boolean;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use the ItemExists property to check whether an element with the specified index exists in the array.

See Also

- Size

Request Support DAC Forum Provide Feedback

5.25.1.1.2.15 ItemIsNull Property(Indexer)

Indicates whether the array element contains a value.

Class

TOraArray

Syntax

```
property ItemIsNull[Index: integer]: boolean;
```

Parameters

Index

Holds the index of the array element.

Remarks

Use IsNull to determine whether the array element with at specific index contains a value. If IsNull is True, the element is empty. If IsNull is False, the element has a value.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.16 ItemSubType Property

Indicates the subtype of an array element.

Class

TOraArray

Syntax

```
property ItemSubType: word;
```

Remarks

Use the ItemSubType property to return the subtype of an array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.17 ItemType Property

Indicates the type of an array element.

Class

TOraArray

Syntax

```
property ItemType: word;
```

Remarks

Use the ItemType property to return the type of an array element.

 Request Support DAC Forum Provide Feedback

5.25.1.1.2.18 MaxSize Property

Defines the maximum number of elements that an array object may hold.

Class

TOraArray

Syntax

```
property MaxSize: integer;
```

Remarks

Use the MaxSize property to get the maximum number of elements that an array object may hold.

See Also

- Size

Request Support DAC Forum Provide Feedback

5.25.1.1.2.19 Size Property

Holds the size of an array.

Class

TOraArray

Syntax

```
property Size: integer;
```

Remarks

Use the Size property to learn the length of an array.

See Also

- ItemExists

5.25.1.1.3 Methods

Methods of the **TOraArray** class.

For a complete list of the **TOraArray** class members, see the TOraArray Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
AppendItem	Appends an element to the end of an array.
Assign	Copies properties and attributes of one object to another one.
Clear	Removes all elements from an array.
CreateObject (inherited from **TOraObject**)	Creates an object.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
InsertItem	Inserts an element at a specific index.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.

MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.

See Also

- TOraArray Class
- TOraArray Class Members

Request Support DAC Forum Provide Feedback

5.25.1.1.3.1 AppendItem Method

Appends an element to the end of an array.

Class

TOraArray

Syntax

```
function AppendItem: integer;
```

Return Value

The index of the appended element.

Remarks

Call the AppendItem function to append an element to the end of an array. Returns the index of the appended element.

See Also

- InsertItem

 Request Support DAC Forum Provide Feedback

5.25.1.1.3.2 Assign Method

Copies properties and attributes of one object to another one.

Class

TOraArray

Syntax

```
procedure Assign(Source: TOraObject); override;
```

Parameters

Source

Holds the source object which properties and attributes will be copied.

Remarks

Call the Assign method to copy the properties and attributes of one object to another one.

See Also

- TOraObject

 Request Support DAC Forum Provide Feedback

5.25.1.1.3.3 Clear Method

Removes all elements from an array.

Class

TOraArray

Syntax

```
procedure Clear;
```

Remarks

Call this method to remove all elements from an array.

5.25.1.1.3.4 InsertItem Method

Inserts an element at a specific index.

Class

TOraArray

Syntax

```
procedure InsertItem(Index: integer);
```

Parameters

Index

Holds the index position.

Remarks

Call the InsertItem procedure to insert an item at a specific index.

See Also

- AppendItem

5.25.1.2 TOraNestTable Class

A class representing a value of the Oracle nested table data type.

For a list of all members of this type, see TOraNestTable members.

Unit

OraObjects

Syntax

```
TOraNestTable = class(TOraArray);
```

Remarks

TOraNestTable represents a value of the Oracle nested table data type. TOraNestTable is inherited from TOraArray. You can get a TOraNestTable object using the TOraDataSet.GetTable method after fetching rows containing array field. Also you can get it using the TOraParam.AsTable method.

Inheritance Hierarchy

TSharedObject
  TDBObject
    TOraObject
      TOraArray
        TOraNestTable

See Also

- TOraArray
- TOraNestedTable
- TOraDataSet.GetTable
- TOraParam.AsTable

 Request Support DAC Forum Provide Feedback

5.25.1.2.1 Members

TOraNestTable class overview.

Properties

Name	Description
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to

	the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ItemAsAnsiString (inherited from **TOraArray**)	Used to read the value of the array element into an

	AnsiString or to assign an AnsiString to the array element.
ItemAsDateTime (inherited from **TOraArray**)	Used to read the value of the array element into a TDateTime variable or to assign a TDateTime variable to the array element.
ItemAsFloat (inherited from **TOraArray**)	Used to read the value of the array element into a Double or to assign a Double to the array element.
ItemAsInteger (inherited from **TOraArray**)	Used to read the value of the array element into an Integer or to assign an Integer to the array element.
ItemAsInterval (inherited from **TOraArray**)	Used to read the value of the array element into a TOraIntervalI variable or to assign a TOraInterval variable to the array element.
ItemAsLargeInt (inherited from **TOraArray**)	Used to read the value of the array element into an Int64 or to assign an Int64 to the array element.
ItemAsLob (inherited from **TOraArray**)	Used to read the value of the array element into a TOraLob variable or to assign a TOraLob variable to the array element.
ItemAsObject (inherited from **TOraArray**)	Used to read the value of the array element into a TOraObject variable or to assign a TOraObject variable to the array element.
ItemAsOCIString (inherited from **TOraArray**)	Used to read the value of the array element into a generic OCIString pointer or to assign the OCIString pointer to the array element.
ItemAsRef (inherited from **TOraArray**)	Used to read the value of the array element into a

	TOraRef variable or to assign a TOraRef variable to the array element.
ItemAsString (inherited from **TOraArray**)	Used to read the value of the array element into a String or to assign a String to the array element.
ItemAsTimeStamp (inherited from **TOraArray**)	Used to read the value of the array element into a TOraTimeStampl variable or to assign a TOraTimeStamp variable to the array element.
ItemAsWideString (inherited from **TOraArray**)	Used to read the value of the array element into a WideString or to assign a WideString to the array element.
ItemExists (inherited from **TOraArray**)	Indicates whether an element with the specified index exists in the array.
ItemIsNull (inherited from **TOraArray**)	Indicates whether the array element contains a value.
ItemSubType (inherited from **TOraArray**)	Indicates the subtype of an array element.
ItemType (inherited from **TOraArray**)	Indicates the type of an array element.
MaxSize (inherited from **TOraArray**)	Defines the maximum number of elements that an array object may hold.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TOraArray**)	Holds the size of an array.

Methods

Name	Description

AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
AppendItem (inherited from **TOraArray**)	Appends an element to the end of an array.
Assign (inherited from **TOraArray**)	Copies properties and attributes of one object to another one.
Clear (inherited from **TOraArray**)	Removes all elements from an array.
CreateObject (inherited from **TOraObject**)	Creates an object.
DeleteItem	Deletes an item pointed by its index.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
InsertItem (inherited from **TOraArray**)	Inserts an element at a specific index.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.

5.25.1.2.2 Methods

Methods of the **TOraNestTable** class.

For a complete list of the **TOraNestTable** class members, see the TOraNestTable Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
AppendItem (inherited from **TOraArray**)	Appends an element to the end of an array.
Assign (inherited from **TOraArray**)	Copies properties and attributes of one object to another one.
Clear (inherited from **TOraArray**)	Removes all elements from an array.
CreateObject (inherited from **TOraObject**)	Creates an object.
DeleteItem	Deletes an item pointed by its index.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
InsertItem (inherited from **TOraArray**)	Inserts an element at a specific index.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.

MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.

See Also

- TOraNestTable Class
- TOraNestTable Class Members

 Request Support DAC Forum Provide Feedback

5.25.1.2.2.1 DeleteItem Method

Deletes an item pointed by its index.

Class

TOraNestTable

Syntax

```
procedure DeleteItem(Index: integer);
```

Parameters

Index

Holds the index of the item to delete.

Remarks

Call the DeleteItem method to delete an item pointed by Index.

 Request Support DAC Forum Provide Feedback

5.25.1.3 TOraObject Class

A class representing the Oracle object data type value.

For a list of all members of this type, see TOraObject members.

Unit

OraObjects

Syntax

```
TOraObject = class(TDBObject);
```

Remarks

TOraObject represents a value of the Oracle object data type. You can get the description of the object type using the ObjectType method. Use the AllocObject method to create an object of a certain type. To access the attribute value use AttrAsInteger, AttrAsString, etc. You can get a TOraObject object using TOraDataSet.GetObject method after fetching rows containing an array field. Also you can get it using the TOraParam.AsObject method.

Example

```
var
MyType : TOraType;
MyObject : TOraObject;
. . .
MyType := TOraType.Create;
MyType.Describe('SCOTT.PERSON');
MyObject := TOraObject.Create(Obj.ObjectType);
MyObject.AttrAsString('Name') := 'Ja';
. . .
```

Inheritance Hierarchy

TSharedObject

  TDBObject

    TOraObject

See Also

- TOraType
- TOraDataSet.GetObject

- TOraParam.AsObject
- TOraRef
- TOraArray
- TOraNestTable

 Request Support DAC Forum Provide Feedback

5.25.1.3.1 Members

TOraObject class overview.

Properties

Name	Description
AttrAsArray	Used to get the TOraArray type attribute value.
AttrAsDateTime	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to

	the contents of an attribute.
AttrAsOCINumber	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull	Used to indicate if the attribute value is NULL.
Indicator	Used to get a pointer to the indicator structure of an object.
Instance	Used to get a pointer to the internal representation of an object.
IsNull	Used to verify if an object is empty.
ObjectType	Used to indicate the object type.
OCISvcCtx	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject	Overloaded. Allocates an object instance.
Assign	Copies properties or other attributes from another

	object.
CreateObject	Creates an object.
Exists	Verifies if an object instance exists in a database.
Flush	Places modifications made to the object to the database.
FreeObject	Deallocates and frees an object instance.
IsDirty	Verifies if an object instance is marked as dirty.
IsLocked	Verifies if an object instance is marked as locked.
Lock	Marks an object as locked for update.
MarkDelete	Marks an object as being deleted.
MarkUpdate	Marks an object as being updated.
Refresh	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark	Marks an object as not being dirty.

Request Support DAC Forum Provide Feedback

5.25.1.3.2 Properties

Properties of the **TOraObject** class.

For a complete list of the **TOraObject** class members, see the TOraObject Members topic.

Public

Name	Description
AttrAsArray	Used to get the TOraArray type attribute value.
AttrAsDateTime	Used to read the attribute's data value into TDateTime,

	or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull	Used to indicate if the attribute value is NULL.
Indicator	Used to get a pointer to the indicator structure of an object.
Instance	Used to get a pointer to the internal representation of an

	object.
IsNull	Used to verify if an object is empty.
ObjectType	Used to indicate the object type.
OCISvcCtx	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

See Also

- TOraObject Class
- TOraObject Class Members

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.1 AttrAsArray Property(Indexer)

Used to get the TOraArray type attribute value.

Class

TOraObject

Syntax

```
property AttrAsArray[const Name: string]: TOraArray;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsArray property to get the value of an attribute of the TOraArray type.

Provide an attribute name in the Name index parameter.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.2 AttrAsDateTime Property(Indexer)

Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsDateTime[const Name: string]: TDateTime;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsDateTime property to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.

Request Support DAC Forum Provide Feedback

5.25.1.3.2.3 AttrAsFloat Property(Indexer)

Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsFloat[const Name: string]: double;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsFloat property to read the attribute's data value into a double, or to assign a

double value to the contents of an attribute.

5.25.1.3.2.4 AttrAsInteger Property(Indexer)

Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsInteger[const Name: string]: integer;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsInteger property to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.

5.25.1.3.2.5 AttrAsLob Property(Indexer)

Used to get reference to the TOraLob object that represents the LOBAttribute value.

Class

TOraObject

Syntax

```
property AttrAsLob[const Name: string]: TOraLob;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsLob property to get reference to the TOraLob object that represents the LOBAttribute value.

See Also

- TOraLob

Request Support DAC Forum Provide Feedback

5.25.1.3.2.6 AttrAsObject Property(Indexer)

Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsObject[const Name: string]: TOraObject;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsObject property to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute. Type of an attribute specified by the Name parameter must be Object. You can use AttrAs... properties to access its attributes after.

Another way to get an attribute value of a nested object is using of the name path.

Example

```
Example 1.
Street1 := MyObject.AttrAsObject('Address').AsString('Street');
Example 1.
Street2 := MyObject.AttrAsString('Address.Street');
```

See Also

- TOraObject

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.7 AttrAsOCIDate Property(Indexer)

Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsOCIDate[const Name: string]: OCIDate;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsOCIDate property to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.8 AttrAsOCINumber Property(Indexer)

Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsOCINumber[const Name: string]: OCINumber;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsOCINumber property to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.9 AttrAsOCIString Property(Indexer)

Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsOCIString[const Name: string]: pOCIString;
```

Parameters

Name

Holds an attribute name.

Remarks

Use AttrAsOCIString property to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.10 AttrAsString Property(Indexer)

Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.

Class

TOraObject

Syntax

```
property AttrAsString[const Name: string]: string;
```

Parameters

Name

Holds an attribute name.

Remarks

Use the AttrAsString property to read the attribute's data value as string, or to assign a string value to the contents of an attribute.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.11 AttrIsNull Property(Indexer)

Used to indicate if the attribute value is NULL.

Class

TOraObject

Syntax

```
property AttrIsNull[const Name: string]: boolean;
```

Parameters

Name

Holds an attribute name.

Remarks

Check the AttrIsNull property to learn whether the attribute value is NULL.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.12 Indicator Property

Used to get a pointer to the indicator structure of an object.

Class

TOraObject

Syntax

```
property Indicator: IntPtr;
```

Remarks

Use the Indicator property to get a pointer to the indicator structure of an object.

Request Support DAC Forum Provide Feedback

5.25.1.3.2.13 Instance Property

Used to get a pointer to the internal representation of an object.

Class

TOraObject

Syntax

```
property Instance: IntPtr;
```

Remarks

Use the Instance property to get a pointer to the internal representation of an object.

Request Support DAC Forum Provide Feedback

5.25.1.3.2.14 IsNull Property

Used to verify if an object is empty.

Class

TOraObject

Syntax

```
property IsNull: boolean;
```

Remarks

Use the IsNull property to verify whether this object is empty. Assign a value to this property to set the object to Null or not Null.

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.15 ObjectType Property

Used to indicate the object type.

Class

TOraObject

Syntax

```
property ObjectType: TOraType;
```

Remarks

Read the ObjectType property to learn the type of an object.

See Also

- TOraType

 Request Support DAC Forum Provide Feedback

5.25.1.3.2.16 OCISvcCtx Property

Used to assign a service context handle.

Class

TOraObject

Syntax

```
property OCISvcCtx: TOCISvcCtx;
```

Remarks

Use the OCISvcCtx property to assign a service context handle. Some operations with objects require a service context handle. To get a service context handle use TOraSession.OCISvcCtx.

See Also

- TOraSession.OCISvcCtx

Request Support DAC Forum Provide Feedback

5.25.1.3.3 Methods

Methods of the **TOraObject** class.

For a complete list of the **TOraObject** class members, see the TOraObject Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject	Overloaded. Allocates an object instance.
Assign	Copies properties or other attributes from another object.
CreateObject	Creates an object.
Exists	Verifies if an object instance exists in a database.
Flush	Places modifications made to the object to the database.
FreeObject	Deallocates and frees an object instance.
IsDirty	Verifies if an object instance is marked as dirty.

IsLocked	Verifies if an object instance is marked as locked.
Lock	Marks an object as locked for update.
MarkDelete	Marks an object as being deleted.
MarkUpdate	Marks an object as being updated.
Refresh	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark	Marks an object as not being dirty.

See Also

- TOraObject Class
- TOraObject Class Members

5.25.1.3.3.1 AllocObject Method

Allocates an object instance.

Class

`TOraObject`

Overload List

Name	Description
AllocObject	Allocates an object instance.
AllocObject(OCISvcCtx: TOCISvcCtx)	Allocates an object instance.
AllocObject(OCISvcCtx: TOCISvcCtx; **const** TypeName: **string**)	Allocates an object instance.
AllocObject(**const** TypeName: **string**)	Allocates an object instance.

Allocates an object instance.

Class

TOraObject

Syntax

```
procedure AllocObject; overload; virtual;
```

Remarks

Call the AllocObject method to allocate an object instance. Overloaded procedures with parameters modify either service context handle or object type properties before allocating the object.

 Request Support DAC Forum Provide Feedback

Allocates an object instance.

Class

TOraObject

Syntax

```
procedure AllocObject(OCISvcCtx: TOCISvcCtx); overload;
```

Parameters

OCISvcCtx

Holds the OCI service context.

Remarks

Call the AllocObject method to allocate an object instance. Overloaded procedures with parameters modify either service context handle or object type properties before allocating the object.

 Request Support DAC Forum Provide Feedback

Allocates an object instance.

Class

TOraObject

Syntax

```
procedure AllocObject(OCISvcCtx: TOCISvcCtx; const TypeName:
string); overload; virtual;
```

Parameters

OCISvcCtx

Holds the OCI service context.

TypeName

Holds the name of an Oracle object type that must be allocated.

Remarks

Call the AllocObject method to allocate an object instance. Overloaded procedures with parameters modify either service context handle or object type properties before allocating the object.

 Request Support DAC Forum Provide Feedback

Allocates an object instance.

Class

TOraObject

Syntax

```
procedure AllocObject(const TypeName: string); overload;
virtual;
```

Parameters

TypeName

Holds the name of an Oracle object type that must be allocated.

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

5.25.1.3.3.2 Assign Method

Copies properties or other attributes from another object.

Class

TOraObject

Syntax

```
procedure Assign(Source: TOraObject); virtual;
```

Parameters

Source

Holds the source object to copy properties or other attributes from.

Remarks

Call the Assign method to copy the properties or other attributes from another object.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.3 CreateObject Method

Creates an object.

Class

TOraObject

Syntax

```
procedure CreateObject(OCISvcCtx: TOCISvcCtx; const TypeName:
string); deprecated;
```

Parameters

SvcCtx

Holds an OCI service context handle.

TypeName

Holds the type of the object to create.

Remarks

Call the CreateObject method to create an object.

Note: This method is obsolete, so it's better to use the AllocObject method instead.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.4 Exists Method

Verifies if an object instance exists in a database.

Class

TOraObject

Syntax

```
function Exists: boolean;
```

Remarks

Call the Exists method to verify whether an instance of an object exists in the database.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.5 Flush Method

Places modifications made to the object to the database.

Class

TOraObject

Syntax

```
procedure Flush;
```

Remarks

Call the Flush method to place modifications made to the object into the database.

See the OCIObjectFlush function description in Oracle references for more detailed description of this method.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.6 FreeObject Method

Deallocates and frees an object instance.

Class

TOraObject

Syntax

```
procedure FreeObject(FreeChild: boolean = True); virtual;
```

Parameters

FreeChild

Holds True, if there is an object to be deallocated and freed.

Remarks

Call the FreeObject method to deallocate and free an object instance.

 Request Support DAC Forum Provide Feedback

5.25.1.3.3.7 IsDirty Method

Verifies if an object instance is marked as dirty.

Class

TOraObject

Syntax

```
function IsDirty: boolean;
```

Return Value

True, if the instance is marked as dirty.

Remarks

Call the IsDirty method to verify whether the object instance is marked as dirty.

The return value is True if the instance is dirty.

 Request Support DAC Forum Provide Feedback

5.25.1.3.3.8 IsLocked Method

Verifies if an object instance is marked as locked.

Class

TOraObject

Syntax

```
function IsLocked: boolean;
```

Return Value

True, if the instance is locked.

Remarks

Call the IsLocked method to verify whether the object instance is marked as locked.

The return value is True if the instance is locked.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.9 Lock Method

Marks an object as locked for update.

Class

TOraObject

Syntax

```
procedure Lock;
```

Remarks

Call the Lock method to mark an object as locked for update.

See the OCIObjectLock function description in the Oracle references for more detailed description of this method.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.10 MarkDelete Method

Marks an object as being deleted.

Class

TOraObject

Syntax

```
procedure MarkDelete;
```

Remarks

Call the MarkDelete method to mark an object as being deleted.

See the OCIObjectMarkDelete function description in the Oracle references for more detailed description of this method.

5.25.1.3.3.11 MarkUpdate Method

Marks an object as being updated.

Class

TOraObject

Syntax

```
procedure MarkUpdate;
```

Remarks

Call MarkUpdate method to mark this object as being updated.

See the OCIObjectMarkUpdate function description in the Oracle references for more detailed description of this method.

Reserved.

5.25.1.3.3.12 Refresh Method

Retrieves the latest database image for the object.

Class

TOraObject

Syntax

```
procedure Refresh;
```

Remarks

Call the Refresh method to retrieve the latest database image for the object.

Request Support DAC Forum Provide Feedback

5.25.1.3.3.13 Unmark Method

Marks an object as not being dirty.

Class

TOraObject

Syntax

```
procedure Unmark;
```

Remarks

Call the Unmark method to mark an object as not being dirty.

See the OCIObjectUnmark function description in the Oracle references for more detailed description of this method.

Request Support DAC Forum Provide Feedback

5.25.1.4 TOraRef Class

A class representing the Oracle reference data type value.

For a list of all members of this type, see TOraRef members.

Unit

OraObjects

Syntax

```
TOraRef = class(TOraObject);
```

Remarks

TOraRef represents a value of Oracle reference data type. TOraRef is inherited from TOraObject. You can get TOraRef object by TOraDataSet.GetRef method after fetching rows containing array field. Also you can get it using TOraParam.AsRef method.

Inheritance Hierarchy

TSharedObject
  TDBObject
    TOraObject
      TOraRef

See Also

- TOraObject
- TOraType
- TOraDataSet.GetRef
- TOraParam.AsRef

5.25.1.4.1 Members

TOraRef class overview.

Properties

Name	Description
AsHex	Used to convert the REF value converted to the hexadecimal string.
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to

	assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCIRef	Used to get or set the OCIRef handle of reference.
OCIRefPtr	Provides reference to OCIRef handle.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
Assign (inherited from **TOraObject**)	Copies properties or other attributes from another object.
Clear	Invalidates reference so that it would no longer point to an object.
CreateObject (inherited from **TOraObject**)	Creates an object.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.

Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Pin	Pins a referenceable object.
RefIsNull	Verifies whether reference is associated with an object and the identifier of that object is currently Null.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.
Unpin	Unpins a referenceable object.

Request Support DAC Forum Provide Feedback

5.25.1.4.2 Properties

Properties of the **TOraRef** class.

For a complete list of the **TOraRef** class members, see the TOraRef Members topic.

Public

Name	Description

AsHex	Used to convert the REF value converted to the hexadecimal string.
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.

AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCIRef	Used to get or set the OCIRef handle of reference.
OCIRefPtr	Provides reference to OCIRef handle.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

See Also

- TOraRef Class
- TOraRef Class Members

Request Support DAC Forum Provide Feedback

5.25.1.4.2.1 AsHex Property

Used to convert the REF value converted to the hexadecimal string.

Class

TOraRef

Syntax

```
property AsHex: string;
```

Remarks

Use the AsHex property to get the REF value converted to hexadecimal string.

See Also

- TOraRef

Request Support DAC Forum Provide Feedback

5.25.1.4.2.2 OCIRef Property

Used to get or set the OCIRef handle of reference.

Class

TOraRef

Syntax

```
property OCIRef: pOCIRef;
```

Remarks

Use the OCIRef property to get or set the OCIRef handle of reference.

See Also

- TOraRef

Request Support DAC Forum Provide Feedback

5.25.1.4.2.3 OCIRefPtr Property

Provides reference to OCIRef handle.

Class

TOraRef

Syntax

```
property OCIRefPtr: ppOCIRef;
```

Remarks

Use the OCIRefPtr property to get reference to OCIRef handle.

See Also

- TOraRef

Request Support DAC Forum Provide Feedback

5.25.1.4.3 Methods

Methods of the **TOraRef** class.

For a complete list of the **TOraRef** class members, see the TOraRef Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject (inherited from **TOraObject**)	Overloaded. Allocates an object instance.
Assign (inherited from **TOraObject**)	Copies properties or other attributes from another object.
Clear	Invalidates reference so that it would no longer point to an object.
CreateObject (inherited from **TOraObject**)	Creates an object.
Exists (inherited from **TOraObject**)	Verifies if an object instance exists in a database.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.

Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Pin	Pins a referenceable object.
RefIsNull	Verifies whether reference is associated with an object and the identifier of that object is currently Null.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.
Unpin	Unpins a referenceable object.

See Also

- TOraRef Class
- TOraRef Class Members

5.25.1.4.3.1 Clear Method

Invalidates reference so that it would no longer point to an object.

Class

TOraRef

Syntax

```
procedure Clear;
```

Remarks

Call the Clear method to invalidate reference so that it would no longer point to an object.

See Also

- TOraRef

 Request Support DAC Forum Provide Feedback

5.25.1.4.3.2 Pin Method

Pins a referenceable object.

Class

TOraRef

Syntax

```
procedure Pin;
```

Remarks

Call the Pin method to pin a referenceable object.

See the OCIObjectPin function description in the Oracle documentation for more information.

See Also

- TOraRef

 Request Support DAC Forum Provide Feedback

5.25.1.4.3.3 RefIsNull Method

Verifies whether reference is associated with an object and the identifier of that object is currently Null.

Class

TOraRef

Syntax

```
function RefIsNull: boolean;
```

Return Value

True, if object identifier is Null. False otherwise.

Remarks

Call the RefIsNull method to verify whether reference is associated with an object and the identifier of that object is currently Null.

The return value is True if object's identifier is Null.

See Also

- TOraRef

Request Support DAC Forum Provide Feedback

5.25.1.4.3.4 Unpin Method

Unpins a referenceable object.

Class

TOraRef

Syntax

```
procedure Unpin;
```

Remarks

Call the Unpin method to unpin a referenceable object.

See the OCIObjectUnpin function description in the Oracle documentation for more information.

See Also

- TOraRef

Request Support DAC Forum Provide Feedback

5.25.1.5 TOraType Class

A class holding information about Oracle type required for TOraObject objects.

For a list of all members of this type, see TOraType members.

Unit

OraObjects

Syntax

```
TOraType = class(TObjectType);
```

Remarks

The TOraType class represents Oracle type and holds information about type required for TOraObject objects.

Inheritance Hierarchy

TSharedObject
  TObjectType
    TOraType

See Also

- TObjectType
- TOraObject

Request Support DAC Forum Provide Feedback

5.25.1.5.1 Members

TOraType class overview.

Properties

Name	Description
AttributeCount (inherited from **TObjectType**)	Used to indicate the number of attributes of type.

Attributes (inherited from **TObjectType**)	Used to access separate attributes.
DataSize	Used to specify the size for allocating an Oracle object in the memory.
DataType (inherited from **TObjectType**)	Used to indicate the type of object dtObject, dtArray or dtTable.
IndicatorSize	Contains the size of the indicator structure for Oracle object.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TObjectType**)	Used to learn the size of an object instance.
TDO	Used to retrieve the type descriptor object for type.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
FindAttribute (inherited from **TObjectType**)	Indicates whether a specified Attribute component is referenced in the TAttributes object.
Release (inherited from **TSharedObject**)	Decrements the reference count.

Request Support DAC Forum Provide Feedback

5.25.1.5.2 Properties

Properties of the **TOraType** class.

For a complete list of the **TOraType** class members, see the TOraType Members topic.

Public

Name	Description
AttributeCount (inherited from **TObjectType**)	Used to indicate the number of attributes of type.
Attributes (inherited from **TObjectType**)	Used to access separate attributes.
DataSize	Used to specify the size for allocating an Oracle object in the memory.
DataType (inherited from **TObjectType**)	Used to indicate the type of object dtObject, dtArray or dtTable.
IndicatorSize	Contains the size of the indicator structure for Oracle object.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.
Size (inherited from **TObjectType**)	Used to learn the size of an object instance.
TDO	Used to retrieve the type descriptor object for type.

See Also

- TOraType Class
- TOraType Class Members

Request Support DAC Forum Provide Feedback

5.25.1.5.2.1 DataSize Property

Used to specify the size for allocating an Oracle object in the memory.

Class

TOraType

Syntax

```
property DataSize: word;
```

Remarks

Use the DataSize property to determine the size for allocating an Oracle object in the memory.

See Also

- IndicatorSize

Request Support DAC Forum Provide Feedback

5.25.1.5.2.2 IndicatorSize Property

Contains the size of the indicator structure for Oracle object.

Class

TOraType

Syntax

```
property IndicatorSize: word;
```

Remarks

Use the IndicatorSize property to learn the size of the indicator structure for Oracle object.

See Also

- DataSize

Request Support DAC Forum Provide Feedback

5.25.1.5.2.3 TDO Property

Used to retrieve the type descriptor object for type.

Class

TOraType

Syntax

```
property TDO: pOCIType;
```

Remarks

Use the TDO property to retrieve the type descriptor object for type.

Request Support DAC Forum Provide Feedback

5.25.1.6 TOraXML Class

A class representing a value of the Oracle SYS.XMLTYPE type.

For a list of all members of this type, see TOraXML members.

Unit

OraObjects

Syntax

```
TOraXML = class(TOraObject);
```

Remarks

TOraXML represents a value of Oracle SYS.XMLTYPE type. Use TOraXML.AllocObject method to create an object. Use TOraXML.AsString and TOraXML.LoadFromStream to initialize XML value. You can get TOraXML object by TOraXMLField.AsXML method after fetching rows contained XMLTYPE field. Also you can get it by TOraParam.AsXML method. To manipulate obtained XML document use TOraXML.Extract, TOraXML.Exists, and TOraXML.Transform functions.

Example

```
var
   XMLDoc : TOraXML;
begin
   XMLDoc := TOraXML.Create();
   XMLDoc.OCISvcCtx := OraSession1.OCISvcCtx;
   XMLDoc.AsString := '<root><node1>value</node1></root>';
...
```

Inheritance Hierarchy

TSharedObject
  TDBObject

TOraObject

TOraXML

See Also

- TOraType
- TOraXMLField

 Request Support DAC Forum Provide Feedback

5.25.1.6.1 Members

TOraXML class overview.

Properties

Name	Description
AsString	Used to get and set the XML document value.
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.
AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.

AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

Methods

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.

AllocObject	Overloaded. Allocates an object instance from an existing TOraLob object.
Assign (inherited from **TOraObject**)	Copies properties or other attributes from another object.
CreateObject (inherited from **TOraObject**)	Creates an object.
Exists	Checks if the given set of nodes in the TOraXML exists.
Extract	Extracts the given set of nodes from the TOraXML.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
GetSchema	Determines the based schema document, schema URL and root element used for the current XMLType creation.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
IsSchemaBased	Determines if an XMLType instance is schema-based.
LoadFromStream	Copies the contents of a stream into the TOraXML object.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToStream	Copies the contents of a

	TOraXML object to a stream.
Transform	Transforms and returns a TOraXML object.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.
Validate	Checks if the input instance conforms to a specified XML schema.

Request Support DAC Forum Provide Feedback

5.25.1.6.2 Properties

Properties of the **TOraXML** class.

For a complete list of the **TOraXML** class members, see the TOraXML Members topic.

Public

Name	Description
AsString	Used to get and set the XML document value.
AttrAsArray (inherited from **TOraObject**)	Used to get the TOraArray type attribute value.
AttrAsDateTime (inherited from **TOraObject**)	Used to read the attribute's data value into TDateTime, or to assign a TDateTime value to the contents of an attribute.
AttrAsFloat (inherited from **TOraObject**)	Used to read the attribute's data value into a double, or to assign a double value to the contents of an attribute.
AttrAsInteger (inherited from **TOraObject**)	Used to read the attribute's data value as integer, or to assign an integer value to the contents of an attribute.
AttrAsLob (inherited from **TOraObject**)	Used to get reference to the TOraLob object that represents the LOBAttribute value.

AttrAsObject (inherited from **TOraObject**)	Used to read the attribute's data value as TOraObject, or to assign a TOraObject value to the contents of an attribute.
AttrAsOCIDate (inherited from **TOraObject**)	Used to read the attribute's data value as OCIDate, or to assign an OCIDate value to the contents of an attribute.
AttrAsOCINumber (inherited from **TOraObject**)	Used to read the attribute's data value into OCINumber, or to assign an OCINumber value to the contents of an attribute.
AttrAsOCIString (inherited from **TOraObject**)	Used to read the attribute's data value as OCIString, or to assign an OCIString value to the contents of an attribute.
AttrAsString (inherited from **TOraObject**)	Used to read the attribute's data value as string, or to assign a string value to the contents of an attribute.
AttrIsNull (inherited from **TOraObject**)	Used to indicate if the attribute value is NULL.
Indicator (inherited from **TOraObject**)	Used to get a pointer to the indicator structure of an object.
Instance (inherited from **TOraObject**)	Used to get a pointer to the internal representation of an object.
IsNull (inherited from **TOraObject**)	Used to verify if an object is empty.
ObjectType (inherited from **TOraObject**)	Used to indicate the object type.
OCISvcCtx (inherited from **TOraObject**)	Used to assign a service context handle.
RefCount (inherited from **TSharedObject**)	Used to return the count of reference to a TSharedObject object.

See Also

- TOraXML Class

- TOraXML Class Members

 Request Support DAC Forum Provide Feedback

5.25.1.6.2.1 AsString Property

Used to get and set the XML document value.

Class

TOraXML

Syntax

```
property AsString: string;
```

Remarks

Use the AsString property to get and set the XML document value. Reading AsString when TOraObject.IsNull is True returns empty string.

See Also

- TOraObject.IsNull

 Request Support DAC Forum Provide Feedback

5.25.1.6.3 Methods

Methods of the **TOraXML** class.

For a complete list of the **TOraXML** class members, see the TOraXML Members topic.

Public

Name	Description
AddRef (inherited from **TSharedObject**)	Increments the reference count for the number of references dependent on the TSharedObject object.
AllocObject	Overloaded. Allocates an object instance from an

	existing TOraLob object.
Assign (inherited from **TOraObject**)	Copies properties or other attributes from another object.
CreateObject (inherited from **TOraObject**)	Creates an object.
Exists	Checks if the given set of nodes in the TOraXML exists.
Extract	Extracts the given set of nodes from the TOraXML.
Flush (inherited from **TOraObject**)	Places modifications made to the object to the database.
FreeObject (inherited from **TOraObject**)	Deallocates and frees an object instance.
GetSchema	Determines the based schema document, schema URL and root element used for the current XMLType creation.
IsDirty (inherited from **TOraObject**)	Verifies if an object instance is marked as dirty.
IsLocked (inherited from **TOraObject**)	Verifies if an object instance is marked as locked.
IsSchemaBased	Determines if an XMLType instance is schema-based.
LoadFromStream	Copies the contents of a stream into the TOraXML object.
Lock (inherited from **TOraObject**)	Marks an object as locked for update.
MarkDelete (inherited from **TOraObject**)	Marks an object as being deleted.
MarkUpdate (inherited from **TOraObject**)	Marks an object as being updated.
Refresh (inherited from **TOraObject**)	Retrieves the latest database image for the object.
Release (inherited from **TSharedObject**)	Decrements the reference count.
SaveToStream	Copies the contents of a TOraXML object to a stream.

Transform	Transforms and returns a TOraXML object.
Unmark (inherited from **TOraObject**)	Marks an object as not being dirty.
Validate	Checks if the input instance conforms to a specified XML schema.

See Also

- TOraXML Class
- TOraXML Class Members

 Request Support DAC Forum Provide Feedback

5.25.1.6.3.1 AllocObject Method

Allocates an object instance from an existing TOraLob object.

Class

TOraXML

Overload List

Name	Description
AllocObject	Allocates an object instance from an existing TOraLob object.
AllocObject(AOCISvcCtx: TOCISvcCtx; AOraLob: TOraLob)	Allocates an object instance from an existing TOraLob object.
AllocObject(AOCISvcCtx: TOCISvcCtx; **const** TypeName: **string**)	Allocates an object instance from an existing TOraLob object.
AllocObject(**const** TypeName: **string**)	Allocates an object instance from an existing TOraLob object.

 Request Support DAC Forum Provide Feedback

Allocates an object instance from an existing TOraLob object.

Class

TOraXML

Syntax

```
procedure AllocObject; overload; override;
```

Remarks

Call the AllocObject method to allocate an object instance from an existing TOraLob object. The procedure modifies the service context handle before allocating the object.

See Also

- TOraObject.AllocObject
- TOraLob

Request Support DAC Forum Provide Feedback

Allocates an object instance from an existing TOraLob object.

Class

TOraXML

Syntax

```
procedure AllocObject(AOCISvcCtx: TOCISvcCtx; AOraLob: TOraLob);
overload;
```

Parameters

AOCISvcCtx

Holds the OCI service context.

AOraLob

Holds a TOraLob object the data from which must be copied to an XML object.

Remarks

Call the AllocObject method to allocate an object instance from an existing TOraLob object. The procedure modifies the service context handle before allocating the object.

Request Support DAC Forum Provide Feedback

Reserved.

Allocates an object instance from an existing TOraLob object.

Class

TOraXML

Syntax

```
procedure AllocObject(AOCISvcCtx: TOCISvcCtx; const TypeName:
string); overload; override;
```

Parameters

AOCISvcCtx

Holds the OCI service context.

TypeName

Holds the name of an Oracle object type that must be allocated.

Remarks

Call the AllocObject method to allocate an object instance from an existing TOraLob object.

The procedure modifies the service context handle before allocating the object.

Request Support DAC Forum Provide Feedback

Allocates an object instance from an existing TOraLob object.

Class

TOraXML

Syntax

```
procedure AllocObject(const TypeName: string); overload;
override;
```

Parameters

TypeName

Holds the name of an Oracle object type that must be allocated.

Remarks

Call the AllocObject method to allocate an object instance from an existing TOraLob object. The procedure modifies the service context handle before allocating the object.

Request Support DAC Forum Provide Feedback

5.25.1.6.3.2 Exists Method

Checks if the given set of nodes in the TOraXML exists.

Class

TOraXML

Syntax

```
function Exists(const XPathExpr: string; const NSmap: string =
''): boolean;
```

Parameters

XPathExpr

Holds the set of nodes in the TOraXML.

NSmap

Holds a namespace map of TOraXML.

Return Value

True if specified nodes exist in TOraXML, False otherwise.

Remarks

Call the Exists method to check if the given set of nodes in TOraXML exists. This set of nodes is specified by the XPathExpr parameter.

Returns True if specified nodes exist in the TOraXML, otherwise, returns False.

Example

```
var
   RetDoc: TOraXML;
   Res: boolean;
begin
...
   Edit;
   TOraXMLField(FieldByName('XMLField')).AsXML.AsString :=
      '<root> '+
      '<x xmlns:edi=''http://ecommerce.org/schema''> '+
      '<b>32.18</b> '+
```

```
        '<edi:price units=''Euro''>32.18</edi:price> '+
        '</x> '+
        '</root>';
    Post;
...
    with TOraXMLField(FieldByName('XMLField')).AsXML do begin
        Res := Exists('//edi:price', 'xmlns:edi=http://ecommerce.org/schema');
        Res := Exists('/root/node1'); //Res = False
    end;
end;
```

See Also

- Extract
- GetSchema
- IsSchemaBased
- Transform

Request Support DAC Forum Provide Feedback

5.25.1.6.3.3 Extract Method

Extracts the given set of nodes from the TOraXML.

Class

TOraXML

Syntax

```
procedure Extract(RetDoc: TOraXML; XPathExpr: string; NSmap:
string = '');
```

Parameters

RetDoc

Holds a destination TOraXML object which holds the operation result.

XPathExpr

Holds an expression that specifies the set of nodes to extract.

NSmap

Holds a namespace map of TOraXML.

Remarks

Call the Extract method to extract the given set of nodes from the TOraXML. This set of

nodes is specified by the XPathExpr expression. The original document remains unchanged. If no nodes match the specified expression, returns NULL document. RetDoc parameter is a destination TOraXML object which holds the operation result. RetDoc object must be created before passing it as a parameter. Use XPathExpr parameter to specify what nodes to search for. The NSmap parameter is a namespace that can be used to identify the mapping of prefix(es) specified in the XPath_string to the corresponding namespace(s). The format is "xmlns=a.com xmlns:b=b.com".

Example

```
var
   RetDoc: TOraXML;
   Str: string;
begin
...
   Edit;
   TOraXMLField(FieldByName('XMLField')).AsXML.AsString :=
      '<root> '+
      '<x xmlns:edi=''http://ecommerce.org/schema''> '+
      '<b>32.18</b> '+
      '<edi:price units=''Euro''>32.18</edi:price> '+
      '</x> '+
      '</root>';
   Post;
...
   RetDoc := TOraXML.Create();
   RetDoc.OCISvcCtx := OraSession1.OCISvcCtx;
   try
      with TOraXMLField(FieldByName('XMLField')).AsXML do begin
         Extract(RetDoc, '//edi:price', 'xmlns:edi=http://ecommerce.org/sche
         Str := RetDoc.AsString;
         Extract(RetDoc, '/root/x/b');
         Str := RetDoc.AsString; // Str = '<b>32.18</b>'
      end;
   finally
      RetDoc.Free;
   end;
end;
```

See Also

- Exists
- GetSchema
- IsSchemaBased
- Transform

5.25.1.6.3.4 GetSchema Method

Determines the based schema document, schema URL and root element used for the current XMLType creation.

Class

TOraXML

Syntax

```
procedure GetSchema(SchemaDoc: TOraXML; var SchemaURL: string;
var RootElem: string);
```

Parameters

SchemaDoc

Holds a destination TOraXML object which holds the operation result.

SchemaURL

Holds the schema URL.

RootElem

Holds the root element declared in schema.

Remarks

Call the GetSchema method for schema based TOraXML objects to determine the based schema document, schema URL and root element used for the current XMLType creation. SchemaDoc parameter is a destination TOraXML object which holds the operation result. SchemaDoc object must be created before passing it as a parameter. Use SchemaURL parameter to return based schema URL used for creating XMLType object. Use RootElem parameter to return root element declared in schema that is represented by XML document.

Example

```
var
   RetDoc: TOraXML;
   ResStr: boolean;
begin
   with OraQuery1 do begin
      RetDoc := TOraXML.Create();
      RetDoc.OCISvcCtx := OraSession1.OCISvcCtx;
      try
         with TOraXMLField(FieldByName('XMLField')).AsXML do begin
```

```
            GetSchema(RetDoc, SchemaURL, RootElem);
            ResStr := RetDoc.AsString;
         end;
      finally
         RetDoc.Free;
      end;
   end;
end;
```

See Also

- Exists
- Extract
- IsSchemaBased
- Transform

5.25.1.6.3.5 IsSchemaBased Method

Determines if an XMLType instance is schema-based.

Class

TOraXML

Syntax

```
function IsSchemaBased: boolean;
```

Return Value

True, if the XML instance is schema-based. False otherwise.

Remarks

Call the IsShemaBased method to determine whether the XMLType instance is schema-based. Returns True or False depending on whether the XMLType instance is schema-based.

See Also

- Exists
- Extract

- GetSchema
- Transform

5.25.1.6.3.6 LoadFromStream Method

Copies the contents of a stream into the TOraXML object.

Class

TOraXML

Syntax

```
procedure LoadFromStream(Stream: TStream);
```

Parameters

Stream

Holds the name of a stream from which the field's value is copied.

Remarks

Call the LoadFromStream method to copy the contents of a stream into the TOraXML object. Specify the stream from which the field's value is copied as the value of the Stream parameter.

See Also

- SaveToStream

5.25.1.6.3.7 SaveToStream Method

Copies the contents of a TOraXML object to a stream.

Class

TOraXML

Syntax

```
procedure SaveToStream(Stream: TStream);
```

Parameters

Stream

Hold the name of a stream to copy the contents of a TOraXML object to.

Remarks

Call the SaveToStream method to copy the contents of a TOraXML object to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

See Also

- LoadFromStream

Request Support DAC Forum Provide Feedback

5.25.1.6.3.8 Transform Method

Transforms and returns a TOraXML object.

Class

TOraXML

Syntax

```
procedure Transform(XSLDoc: TOraXML; RetDoc: TOraXML);
```

Parameters

XSLDoc

Holds an XSL document.

RetDoc

Holds a destination TOraXML object which holds the new (transformed) XML document.

Remarks

Call the Transform method to transform and return TOraXML, using the given XSL document in XSLDoc parameter. The new (transformed) XML document is assigned to RetDoc.

XSLDoc parameter is the XSL document to be applied to the XMLType. RetDoc parameter is a destination TOraXML object which holds the operation result. RetDoc object must be created before passing it as a parameter.

Example

```
var
   RetDoc, XSLDoc: TOraXML;
begin
   RetDoc := TOraXML.Create();
   RetDoc.OCISvcCtx := OraSession1.OCISvcCtx;
   XSLDoc := TOraXML.Create();
   XSLDoc.OCISvcCtx := OraSession1.OCISvcCtx;
   try
      with TOraXMLField(FieldByName('XMLField')).AsXML do begin
         XSLDoc.AsString:=XSLDocument;
         Transform(XSLDoc, RetDoc);
         Str := RetDoc.AsString;
      end;
   finally
      RetDoc.Free;
      XSLDoc.Free;
   end;
end;
```

See Also

- Exists
- Extract
- GetSchema
- IsSchemaBased

Request Support DAC Forum Provide Feedback

5.25.1.6.3.9 Validate Method

Checks if the input instance conforms to a specified XML schema.

Class

TOraXML

Syntax

```
function Validate(const SchemaURL: string): boolean;
```

Parameters

SchemaURL

Holds the URL of the XML Schema against which to check the conformance.

Return Value

True, if the input instance conforms to a specified XML schema. False otherwise.

Remarks

Call the Validate method to check if the input instance conforms to a specified XML schema. SchemaURL parameter is the URL of the XML Schema against which to check the conformance.

Example

```
Result := TOraXMLField(FieldByName('XMLField')).AsXML.Validate('http://www.o
```

See Also

- Extract
- GetSchema
- IsSchemaBased
- Transform

Request Support DAC Forum Provide Feedback

5.26 OraPackage

This unit contains implementation of the TOraPackage component.

Classes

Name	Description
TCustomOraPackage	A a base class for components that provide access to packages stored in an Oracle database.
TOraPackage	A component providing access to packages stored in an Oracle database.

Request Support DAC Forum Provide Feedback

5.26.1 Classes

Classes in the **OraPackage** unit.

Classes

Name	Description
TCustomOraPackage	A a base class for components that provide access to packages stored in an Oracle database.
TOraPackage	A component providing access to packages stored in an Oracle database.

Request Support DAC Forum Provide Feedback

5.26.1.1 TCustomOraPackage Class

A a base class for components that provide access to packages stored in an Oracle database.

For a list of all members of this type, see TCustomOraPackage members.

Unit

OraPackage

Syntax

```
TCustomOraPackage = class(TCustomOraComponent);
```

Inheritance Hierarchy

TCustomOraComponent

TCustomOraPackage

Request Support DAC Forum Provide Feedback

5.26.1.1.1 Members

TCustomOraPackage class overview.

Properties

Name	Description
Params	Used to receive values that are the output parameters of package stored procedures.
Session	Used to specify the session component which is associated with this TOraPackage object.

Methods

Name	Description
ExecProc	Calls the stored procedures defined for a given package.
ExecProcEx	Calls the stored procedures defined for a given package.
VariableByName	Provides access to package variables.

5.26.1.1.2 Properties

Properties of the **TCustomOraPackage** class.

For a complete list of the **TCustomOraPackage** class members, see the TCustomOraPackage Members topic.

Public

Name	Description
Params	Used to receive values that are the output parameters of package stored procedures.

Published

Name	Description
Session	Used to specify the session component which is associated with this TOraPackage object.

See Also

- TCustomOraPackage Class
- TCustomOraPackage Class Members

 Request Support DAC Forum Provide Feedback

5.26.1.1.2.1 Params Property

Used to receive values that are the output parameters of package stored procedures.

Class

TCustomOraPackage

Syntax

```
property Params: TOraParams;
```

Remarks

Use the Params property to receive values that are the output parameters of package stored procedures.

 Request Support DAC Forum Provide Feedback

5.26.1.1.2.2 Session Property

Used to specify the session component which is associated with this TOraPackage object.

Class

TCustomOraPackage

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session propery to specify the session component which is associated with this TOraPackage object.

At design-time select a session instance from a dropdown listbox.

 Request Support DAC Forum Provide Feedback

5.26.1.1.3 Methods

Methods of the **TCustomOraPackage** class.

For a complete list of the **TCustomOraPackage** class members, see the TCustomOraPackage Members topic.

Public

Name	Description
ExecProc	Calls the stored procedures defined for a given package.
ExecProcEx	Calls the stored procedures defined for a given package.
VariableByName	Provides access to package variables.

See Also

- TCustomOraPackage Class
- TCustomOraPackage Class Members

 Request Support DAC Forum Provide Feedback

5.26.1.1.3.1 ExecProc Method

Calls the stored procedures defined for a given package.

Class

TCustomOraPackage

Syntax

```
procedure ExecProc(const Name: string); overload;function
ExecProc(Name: string; const Params: array of variant): variant;
overload;
```

Parameters

Name

Holds the name of the stored procedure.

Params

holds the parameter values array.

Return Value

a result, if a stored procedure is a function, Null otherwise.

Remarks

Call the ExecProc method to call the stored procedures defined for a given package. The Name parameter is a name of a stored procedure.

If a stored procedure accepts or returns parameters they must be supplied in the Params array in exactly the same order and number as they appear in the declaration of this stored procedure. If the value for an input parameter was not included in the Params array, the parameter default value is taken. Only the parameter values at the end of the list may be unincluded to the parameter values array. If the parameter has no default value, the NULL value is sent.

For example, the following is a stored procedure declaration in package DBMS_ALERT:

```
DBMS_ALERT.SIGNAL (
    name        IN   VARCHAR2,
    message    IN   VARCHAR2);
```

it may be called with this code:

```
MyOraPackage.ExecProc('SIGNAL', ['MySignalName', 'MyMessage']);
```

This is different from the way ExecProcEx is used.

Note: Stored functions unlike stored procedures return result values. To understand the parameters usage see TCustomDAConnection.ExecProc.

See Also

- ExecProcEx
- TCustomDAConnection.ExecProc
- TCustomDAConnection.ExecSQL
- TCustomDAConnection.ExecSQLEx

Request Support DAC Forum Provide Feedback

5.26.1.1.3.2 ExecProcEx Method

Calls the stored procedures defined for a given package.

Class

TCustomOraPackage

Syntax

```
function ExecProcEx(Name: string; const Params: array of variant): variant;
```

Parameters

Name

Holds the name of the stored procedure.

Params

Holds the parameter values array.

Return Value

a result, if a stored procedure is a function, Null otherwise.

Remarks

Call the ExecProcEx method to call stored procedures defined for a given package. Name parameter is a name of a stored procedure.

If the sored procedure accepts or returns parameters, they must be supplied in the Params array as pairs of parameters' names and values so that every value would come immediately after its name. If the value for an input parameter was not included in the Params array, parameter default value is taken. If the parameter has no default value, NULL value is sent.

For example, the following is a stored procedure declaration in package DBMS_ALERT:

```
DBMS_ALERT.SIGNAL (
```

```
    name        IN   VARCHAR2,
    message     IN   VARCHAR2);
```

it may be called by this code:

```
MyOraPackage.ExecProcEx("SIGNAL", ["name", "MySignalName", "message", "MyMes
```

It is different from the way ExecProc is used.

See Also

- ExecProc
- TCustomDAConnection.ExecProcEx
- TCustomDAConnection.ExecSQLEx

5.26.1.1.3.3 VariableByName Method

Provides access to package variables.

Class

TCustomOraPackage

Syntax

```
function VariableByName(Name: string): TVariable;
```

Parameters

Name

Holds the name of a variable.

Return Value

a reference for a TVariable object.

Remarks

Call the VariableByName method to get access to package variables.

Reference for a variable with the name not defined yet will prepare a placeholder for it.

5.26.1.2 TOraPackage Class

A component providing access to packages stored in an Oracle database.

For a list of all members of this type, see TOraPackage members.

Unit

OraPackage

Syntax

```
TOraPackage = class(TCustomOraPackage);
```

Remarks

Use the TOraPackage components to get access to packages stored in Oracle database.

Packages may encapsulate sets of procedures and functions along with related variables and constants. To get the list of Oracle supplied packages refer to the Oracle online documents.

Inheritance Hierarchy

TCustomOraComponent

TCustomOraPackage

TOraPackage

Request Support DAC Forum Provide Feedback

5.26.1.2.1 Members

TOraPackage class overview.

Properties

Name	Description
PackageName	Used to supply an Oracle package name.
Params (inherited from **TCustomOraPackage**)	Used to receive values that are the output parameters of package stored procedures.
Session (inherited from **TCustomOraPackage**)	Used to specify the session component which is

	associated with this TOraPackage object.

Methods

Name	Description
ExecProc (inherited from **TCustomOraPackage**)	Calls the stored procedures defined for a given package.
ExecProcEx (inherited from **TCustomOraPackage**)	Calls the stored procedures defined for a given package.
VariableByName (inherited from **TCustomOraPackage**)	Provides access to package variables.

Request Support DAC Forum Provide Feedback

5.26.1.2.2 Properties

Properties of the **TOraPackage** class.

For a complete list of the **TOraPackage** class members, see the TOraPackage Members topic.

Public

Name	Description
Params (inherited from **TCustomOraPackage**)	Used to receive values that are the output parameters of package stored procedures.

Published

Name	Description
PackageName	Used to supply an Oracle package name.
Session (inherited from **TCustomOraPackage**)	Used to specify the session component which is associated with this TOraPackage object.

See Also

- TOraPackage Class
- TOraPackage Class Members

 Request Support DAC Forum Provide Feedback

5.26.1.2.2.1 PackageName Property

Used to supply an Oracle package name.

Class

TOraPackage

Syntax

```
property PackageName: string;
```

Remarks

Use the PackageName property to supply an Oracle package name. Subsequent accesses to package methods and variables will be associated with this name.

At design-time dropdown listbox provides all Oracle supplied and user-defined packages to select from.

 Request Support DAC Forum Provide Feedback

5.27 OraProvider

This unit contains implementation of the TOraProvider component.

Classes

Name	Description
TOraProvider	A component for loading data to and from a dataset.

 Request Support DAC Forum Provide Feedback

5.27.1 Classes

Classes in the **OraProvider** unit.

Classes

Name	Description
TOraProvider	A component for loading data to and from a dataset.

 Request Support DAC Forum Provide Feedback

5.27.1.1 TOraProvider Class

A component for loading data to and from a dataset.

For a list of all members of this type, see TOraProvider members.

Unit

OraProvider

Syntax

```
TOraProvider = class(TDataSetProvider);
```

Remarks

TOraProvider provides data to and applies updates from a client dataset. TOraProvider is derived from TDataSetProvider and has same methods and properties as TProvider component.

Note: TOraProvider component works with Delphi (C++Builder) Enterprise edition only. So to install it you need to compile and install OraProvider package **oraprovXX.bpk** .

 Request Support DAC Forum Provide Feedback

5.27.1.1.1 Members

TOraProvider class overview.

 Request Support DAC Forum Provide Feedback

5.28 OraScript

This unit contains implementation of the TOraScript component.

Classes

Name	Description
TOraScript	A component for executing several SQL statements one by one.
TOraStatement	A class used for controlling single SQL statements of the script.
TOraStatements	A class for holding a collection of TOraStatement objects.

Request Support DAC Forum Provide Feedback

5.28.1 Classes

Classes in the **OraScript** unit.

Classes

Name	Description
TOraScript	A component for executing several SQL statements one by one.
TOraStatement	A class used for controlling single SQL statements of the script.
TOraStatements	A class for holding a collection of TOraStatement objects.

Request Support DAC Forum Provide Feedback

5.28.1.1 TOraScript Class

A component for executing several SQL statements one by one.

For a list of all members of this type, see TOraScript members.

Unit

OraScript

Syntax

```
TOraScript = class(TDAScript);
```

Remarks

Often it is necessary to execute several SQL statements one by one. Known way is using a lot of components such as TOraSQL. Usually it is not a good solution. Sometimes it can be performed by anonymous PL/SQL block. But sometimes it does not work. For example, DDL statements cannot be used in PL/SQL. With only one TOraScript component you can execute several SQL statements as one. This sequence of statements is named script. To separate single statements use semicolon (;), slash (/) ,and for PL/SQL - only slash . Note that slash must be the first character in line.

Errors that occur while execution can be processed in the TDAScript.OnError event handler. By default, on error TOraScript shows exception and continues execution.

Inheritance Hierarchy

TDAScript
  TOraScript

See Also

- TOraSQL

5.28.1.1.1 Members

TOraScript class overview.

Properties

Name	Description
Connection (inherited from **TDAScript**)	Used to specify the connection in which the script will be executed.
DataSet	Used to assign a component that will be used by TOraScript to execute statements, and obtain results of execution.
Debug (inherited from **TDAScript**)	Used to display the script execution and all its parameter values.
Delimiter (inherited from **TDAScript**)	Used to set the delimiter string that separates script statements.
EndLine (inherited from **TDAScript**)	Used to get the current statement last line number in a script.
EndOffset (inherited from **TDAScript**)	Used to get the offset in the last line of the current statement.
EndPos (inherited from **TDAScript**)	Used to get the end position of the current statement.
Macros (inherited from **TDAScript**)	Used to change SQL script text in design- or run-time easily.
Session	Used to specify the session in which the script will be executed.
SQL (inherited from **TDAScript**)	Used to get or set script text.
StartLine (inherited from **TDAScript**)	Used to get the current statement start line number in a script.
StartOffset (inherited from **TDAScript**)	Used to get the offset in the first line of the current statement.
StartPos (inherited from **TDAScript**)	Used to get the start position of the current statement in a script.
Statements	Contains the list of statements obtained from

	the SQL property.

Methods

Name	Description
BreakExec (inherited from **TDAScript**)	Stops script execution.
ErrorOffset (inherited from **TDAScript**)	Used to get the offset of the statement if the Execute method raised an exception.
Execute (inherited from **TDAScript**)	Executes a script.
ExecuteFile (inherited from **TDAScript**)	Executes SQL statements contained in a file.
ExecuteNext (inherited from **TDAScript**)	Executes the next statement in the script and then stops.
ExecuteStream (inherited from **TDAScript**)	Executes SQL statements contained in a stream object.
FindMacro (inherited from **TDAScript**)	Finds a macro with the specified name.
MacroByName (inherited from **TDAScript**)	Finds a macro with the specified name.

Events

Name	Description
AfterExecute (inherited from **TDAScript**)	Occurs after a SQL script execution.
BeforeExecute (inherited from **TDAScript**)	Occurs when taking a specific action before executing the current SQL statement is needed.
OnError (inherited from **TDAScript**)	Occurs when Oracle raises an error.

5.28.1.1.2 Properties

Properties of the **TOraScript** class.

For a complete list of the **TOraScript** class members, see the TOraScript Members topic.

Public

Name	Description
Connection (inherited from **TDAScript**)	Used to specify the connection in which the script will be executed.
EndLine (inherited from **TDAScript**)	Used to get the current statement last line number in a script.
EndOffset (inherited from **TDAScript**)	Used to get the offset in the last line of the current statement.
EndPos (inherited from **TDAScript**)	Used to get the end position of the current statement.
StartLine (inherited from **TDAScript**)	Used to get the current statement start line number in a script.
StartOffset (inherited from **TDAScript**)	Used to get the offset in the first line of the current statement.
StartPos (inherited from **TDAScript**)	Used to get the start position of the current statement in a script.
Statements	Contains the list of statements obtained from the SQL property.

Published

Name	Description
DataSet	Used to assign a component that will be used by TOraScript to execute statements, and obtain results of execution.
Debug (inherited from **TDAScript**)	Used to display the script execution and all its parameter values.

Delimiter (inherited from **TDAScript**)	Used to set the delimiter string that separates script statements.
Macros (inherited from **TDAScript**)	Used to change SQL script text in design- or run-time easily.
Session	Used to specify the session in which the script will be executed.
SQL (inherited from **TDAScript**)	Used to get or set script text.

See Also

- TOraScript Class
- TOraScript Class Members

 Request Support DAC Forum Provide Feedback

5.28.1.1.2.1 DataSet Property

Used to assign a component that will be used by TOraScript to execute statements, and obtain results of execution.

Class

TOraScript

Syntax

```
property DataSet: TOraDataSet;
```

See Also

- TOraDataSet
- TDAScript.ExecuteNext

 Request Support DAC Forum Provide Feedback

5.28.1.1.2.2 Session Property

Used to specify the session in which the script will be executed.

Class

TOraScript

Syntax

```
property Session: TOraSession;
```

Remarks

Use the Session property to specify the session in which the script will be executed. If Session is not connected, Execute method calls Session.Connect.

See Also

- TOraSession

5.28.1.1.2.3 Statements Property

Contains the list of statements obtained from the SQL property.

Class

TOraScript

Syntax

```
property Statements: TOraStatements;
```

Remarks

Contains the list of statements that are obtained from the SQL property. Access the Statements property to view SQL statement, set parameters or execute the specified statement. Statements is a zero-based array of statement records. Index specifies the array element to access.

For example, consider the following script:

```
CREATE TABLE A (FIELD1 INTEGER);
INSERT INTO A VALUES(1);
INSERT INTO A VALUES(2);
INSERT INTO A VALUES(3);
CREATE TABLE B (FIELD1 INTEGER);
INSERT INTO B VALUES(1);
INSERT INTO B VALUES(2);
INSERT INTO B VALUES(3);
```

Note: The list of statements is created and filled when the value of the Statements property is requested for the first time. That's why the first access to the Statements property can take a long time.

Example

You can use the Statements property as presented below:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i: integer;
begin
   with Script do
   begin
      for i := 0 to Statements.Count - 1 do
         if Copy(Statements[i].SQL, 1, 6) <> 'CREATE' then
            Statements[i].Execute;
   end;
end;
```

See Also

- TOraStatements

5.28.1.2 TOraStatement Class

A class used for controlling single SQL statements of the script.

For a list of all members of this type, see TOraStatement members.

Unit

OraScript

Syntax

```
TOraStatement = class(TDAStatement);
```

Remarks

TOraScript contains SQL statements, represented as TOraStatement objects. TOraStatement class has attributes and methods for controlling single SQL statements of the script.

Inheritance Hierarchy

TDAStatement

TOraStatement

See Also

- TOraScript
- TOraStatements
- TOraStatements

Request Support DAC Forum Provide Feedback

5.28.1.2.1 Members

TOraStatement class overview.

Properties

Name	Description
EndLine (inherited from **TDAStatement**)	Used to determine the number of the last statement line in a script.
EndOffset (inherited from **TDAStatement**)	Used to get the offset in the last line of the statement.
EndPos (inherited from **TDAStatement**)	Used to get the end position of the statement in a script.
Omit (inherited from **TDAStatement**)	Used to avoid execution of a statement.
Params	Contains parameters for an SQL statement.
Script (inherited from **TDAStatement**)	Used to determine the TDAScript object the SQL Statement belongs to.

SQL (inherited from **TDAStatement**)	Used to get or set the text of an SQL statement.
StartLine (inherited from **TDAStatement**)	Used to determine the number of the first statement line in a script.
StartOffset (inherited from **TDAStatement**)	Used to get the offset in the first line of a statement.
StartPos (inherited from **TDAStatement**)	Used to get the start position of the statement in a script.

Methods

Name	Description
Execute (inherited from **TDAStatement**)	Executes a statement.

Request Support DAC Forum Provide Feedback

5.28.1.2.2 Properties

Properties of the **TOraStatement** class.

For a complete list of the **TOraStatement** class members, see the TOraStatement Members topic.

Public

Name	Description
EndLine (inherited from **TDAStatement**)	Used to determine the number of the last statement line in a script.
EndOffset (inherited from **TDAStatement**)	Used to get the offset in the last line of the statement.
EndPos (inherited from **TDAStatement**)	Used to get the end position of the statement in a script.
Omit (inherited from **TDAStatement**)	Used to avoid execution of a statement.
Params	Contains parameters for an SQL statement.
Script (inherited from **TDAStatement**)	Used to determine the TDAScript object the SQL Statement belongs to.

SQL (inherited from **TDAStatement**)	Used to get or set the text of an SQL statement.
StartLine (inherited from **TDAStatement**)	Used to determine the number of the first statement line in a script.
StartOffset (inherited from **TDAStatement**)	Used to get the offset in the first line of a statement.
StartPos (inherited from **TDAStatement**)	Used to get the start position of the statement in a script.

See Also

- TOraStatement Class
- TOraStatement Class Members

Request Support DAC Forum Provide Feedback

5.28.1.2.2.1 Params Property

Contains parameters for an SQL statement.

Class

TOraStatement

Syntax

```
property Params: TOraParams;
```

Remarks

The Params property contains parameters for an SQL statement.

Access Params at runtime to view and set parameter names, values, and data types dynamically. Params is a zero-based array of parameter records. Index specifies the array element to access.

See Also

- TOraParam

Request Support DAC Forum Provide Feedback

Reserved.

5.28.1.3 TOraStatements Class

A class for holding a collection of TOraStatement objects.

For a list of all members of this type, see TOraStatements members.

Unit

OraScript

Syntax

```
TOraStatements = class(TDAStatements);
```

Remarks

Each TOraStatements holds a collection of TOraStatement objects. TOraStatements maintains an index of the statements in its TOraStatements.Items array. The Count property contains the number of statements in the collection. Use TOraStatements class to manipulate script SQL statements.

Inheritance Hierarchy

TDAStatements
  TOraStatements

See Also

- TDAScript
- TDAStatement

Request Support DAC Forum Provide Feedback

5.28.1.3.1 Members

TOraStatements class overview.

Properties

Name	Description

Items	Used to access individual script statements.

 Request Support DAC Forum Provide Feedback

5.28.1.3.2 Properties

Properties of the **TOraStatements** class.

For a complete list of the **TOraStatements** class members, see the TOraStatements Members topic.

Public

Name	Description
Items	Used to access individual script statements.

See Also

- TOraStatements Class
- TOraStatements Class Members

 Request Support DAC Forum Provide Feedback

5.28.1.3.2.1 Items Property(Indexer)

Used to access individual script statements.

Class

TOraStatements

Syntax

```
property Items[Index: Integer]: TOraStatement; default;
```

Parameters

Index

Holds an array of the statements index.

Remarks

Use the Items property to access individual script statements. The value of the Index parameter corresponds to the Index property of TOraStatement.

See Also

- TOraStatement

 Request Support DAC Forum Provide Feedback

5.29 OraServices

5.29.1 Constants

Constants in the **OraServices** unit.

Constants

Name	Description
OraDataTypeMap	This unit contains the implementation of mapping between Oracle and Delphi data types.

 Request Support DAC Forum Provide Feedback

5.29.1.1 OraDataTypeMap Constant

This unit contains the implementation of mapping between Oracle and Delphi data types.

Constants

Name	Description
oraAnyData	Used to map **ANYDATA** to Delphi data types.
oraBFile	Used to map **BFILE** to Delphi data types.
oraBinaryDouble	Used to map **BINARY_DOUBLE** to Delphi data types.

oraBinaryFloat	Used to map **BINARY_FLOAT** to Delphi data types.
oraBlob	Used to map **BLOB** to Delphi data types.
oraCFile	Used to map **CFILE** to Delphi data types.
oraChar	Used to map **CHAR** to Delphi data types.
oraClob	Used to map **CLOB** to Delphi data types.
oraCursor	Used to map **CURSOR** to Delphi data types.
oraDate	Used to map **DATE** to Delphi data types.
oraDoublePrecision	Used to map **DOUBLE PRECISION** to Delphi data types.
oraFloat	Used to map **FLOAT** to Delphi data types.
oraInteger	Used to map **INTEGER** to Delphi data types.
oraIntervalDS	Used to map **INTERVAL DAY TO SECOND** to Delphi data types.
oraIntervalYM	Used to map **INTERVAL YEAR TO MONTH** to Delphi data types.
oraLabel	Used to map **MLSLABEL** to Delphi data types.
oraLong	Used to map **LONG** to Delphi data types.
oraLongRaw	Used to map **LONG RAW** to Delphi data types.
oraNChar	Used to map **NCHAR** to Delphi data types.
oraNClob	Used to map **NCLOB** to Delphi data types.
oraNumber	Used to map **NUMBER** to Delphi data types.
oraNVarchar2	Used to map **NVARCHAR2** to Delphi data types.

oraObject	Used to map **OBJECT** to Delphi data types.
oraRaw	Used to map **RAW** to Delphi data types.
oraReference	Used to map **REF** to Delphi data types.
oraRowID	Used to map **ROWID** to Delphi data types.
oraTimeStamp	Used to map **TIMESTAMP** to Delphi data types.
oraTimeStampWithLocalTimeZone	Used to map **TIMESTAMP WITH LOCAL TIME ZONE** to Delphi data types.
oraTimeStampWithTimeZone	Used to map **TIMESTAMP WITH TIME ZONE** to Delphi data types.
oraUndefined	Used to map **UNDEFINED** to Delphi data types.
oraURowID	Used to map **UROWID** to Delphi data types.
oraVarchar2	Used to map **VARCHAR2** to Delphi data types.
oraXML	Used to map **XML** to Delphi data types.

Request Support DAC Forum Provide Feedback

5.30 OraSmart

This unit contains the TSmartQuery and TOraTable components.

Classes

Name	Description
TCustomSmartQuery	A base class defining functionality for descendant classes that access database using dynamically generated SQL statements.
TOraTable	A component for retrieving and updating data in a

	single table without writing SQL statements.
TSmartQuery	A component providing TCustomSmartQuery.Expand fields feature, that lets all data controls be aware of all the fields belonging to updating table an not only those requested in SELECT clause, and TCustomSmartQuery.SmartRefresh feature (in Professional and Developer editions only).
TSmartQueryOptions	This class allows setting up the behaviour of the TSmartQuery class.

Enumerations

Name	Description
TSmartState	Specifies if TCustomSmartQuery is in view mode.

Request Support DAC Forum Provide Feedback

5.30.1 Classes

Classes in the **OraSmart** unit.

Classes

Name	Description
TCustomSmartQuery	A base class defining functionality for descendant classes that access database using dynamically generated SQL statements.
TOraTable	A component for retrieving and updating data in a single table without writing

	SQL statements.
TSmartQuery	A component providing TCustomSmartQuery.Expand fields feature, that lets all data controls be aware of all the fields belonging to updating table an not only those requested in SELECT clause, and TCustomSmartQuery.SmartRefresh feature (in Professional and Developer editions only).
TSmartQueryOptions	This class allows setting up the behaviour of the TSmartQuery class.

Request Support DAC Forum Provide Feedback

5.30.1.1 TCustomSmartQuery Class

A base class defining functionality for descendant classes that access database using dynamically generated SQL statements.

For a list of all members of this type, see TCustomSmartQuery members.

Unit

OraSmart

Syntax

```
TCustomSmartQuery = class(TCustomOraQuery);
```

Remarks

TCustomSmartQuery is a base class that defines functionality for descendant classes that access database using dynamically generated SQL statements. Applications never use TCustomSmartQuery objects directly. Instead they use descendants of TCustomSmartQuery, such as TSmartQuery and TOraTable.

TSmartQuery is an alternative to TOraQuery. It provides TCustomSmartQuery.Expand fields

feature, that lets all data controls be aware of all the fields belonging to updating table and not only those requested in a SELECT clause, and TCustomSmartQuery.SmartRefresh feature (in Professional and Developer editions only).

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet
      TCustomOraQuery
        TCustomSmartQuery

See Also

- TOraQuery
- TSmartQuery
- TOraStoredProc
- TOraTable

Request Support DAC Forum Provide Feedback

5.30.1.1.1 Members

TCustomSmartQuery class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.

CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DependEvents	Used to get or set the names of the events that dataset will depend on in TCustomSmartQuery.SmartRefresh mode.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
Expand	Lets all data controls be aware of all the fields belonging to updating table.

FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode (inherited from **TOraDataSet**)	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.

Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshEvent	Used to get or set the name of the event that dataset will be waiting for in TCustomSmartQuery.SmartRefresh mode.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SmartRefresh	Let TCustomSmartQuery components work in a concurrent environment.
SmartState	Defines if TCustomSmartQuery is in view mode.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.

SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record

	when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.

EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and

	accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record

	and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent

	assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.
View	Allows viewing all fields of the updating table.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterSmartRefresh	Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.
AfterUpdateExecute (inherited from	Occurs after executing insert, delete, update, lock

TCustomDADataSet)	and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.
RefreshFields	Occures before an application posts changes for the current record to the database or cache.

Request Support DAC Forum Provide Feedback

5.30.1.1.2 Properties

Properties of the **TCustomSmartQuery** class.

For a complete list of the **TCustomSmartQuery** class members, see the TCustomSmartQuery Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.

CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DependEvents	Used to get or set the names of the events that dataset will depend on in TCustomSmartQuery.SmartRefresh mode.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
Expand	Lets all data controls be aware of all the fields belonging to updating table.

FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.

LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode (inherited from **TOraDataSet**)	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.

Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshEvent	Used to get or set the name of the event that dataset will be waiting for in TCustomSmartQuery.SmartRefresh mode.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SmartRefresh	Let TCustomSmartQuery components work in a concurrent environment.
SmartState	Defines if TCustomSmartQuery is in view mode.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.

SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record

	when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

See Also

- TCustomSmartQuery Class
- TCustomSmartQuery Class Members

Request Support DAC Forum Provide Feedback

5.30.1.1.2.1 DependEvents Property

Used to get or set the names of the events that dataset will depend on in SmartRefresh mode.

Class

TCustomSmartQuery

Syntax

```
property DependEvents: string;
```

Remarks

Use DependEvents property to get or set the names of the events that dataset will depend on in SmartRefresh mode. In the other words, when some TCustomsSmartQuery object receives RefreshEvent, it causes an update of every dataset that has received event name in DependEvents list, i.e. all depending datasets are updated.

See Also

- SmartRefresh
- RefreshEvent

Request Support DAC Forum Provide Feedback

5.30.1.1.2.2 Expand Property

Lets all data controls be aware of all the fields belonging to updating table.

Class

TCustomSmartQuery

Syntax

```
property Expand: boolean default False;
```

Remarks

Set Expand property to True to let all data controls be aware of all the fields belonging to updating table an not only those requested in the SELECT clause. Those fields which are not requested explicitly will show no data until the dataset enters edit state for a particular record. At that time all fields for that record will be populated from the database.

Preferred design for an application may include DBGrid component which hides unwanted fields and other data components which reveal those fields only for the selected record. This way large tables with lots of fields are edited by the user only on demand during the course of a session. Thus network traffic may be lessened and memory usage lowered for transfers of the requested fields only.

If Expand property is True you can point in a SELECT statement the fields that will be displayed only in the grid. The rest of the fields from the updating table will be fetched before edit.

Note: This property is mutually exclusive with CachedUpdates property. Thus setting Expand to True leaves CachedUpdates property set to False. Also do not try to read blob fields when they are not expanded, this will cause an exception.

The default value is False.

5.30.1.1.2.3 Options Property

Used to specify the behaviour of TCustomSmartQuery object.

Class

TCustomSmartQuery

Syntax

```
property Options: TSmartQueryOptions;
```

Remarks

Set properties of Options to specify the behaviour of a TCustomSmartQuery object.

See Also

- TCustomDADataSet.Options
- TOraDataSet.Options

Request Support DAC Forum Provide Feedback

5.30.1.1.2.4 RefreshEvent Property

Used to get or set the name of the event that dataset will be waiting for in SmartRefresh mode.

Class

TCustomSmartQuery

Syntax

```
property RefreshEvent: string;
```

Remarks

Use RefreshEvent property to get or set the name of the event that dataset will be waiting for in the SmartRefresh mode. When this event occures, dataset performs the Refresh procedure.

See Also

- SmartRefresh
- DependEvents

 Request Support DAC Forum Provide Feedback

5.30.1.1.2.5 SmartRefresh Property

Let TCustomSmartQuery components work in a concurrent environment.

Class

TCustomSmartQuery

Syntax

```
property SmartRefresh: boolean default False;
```

Remarks

Set SmartRefresh property to True to let TCustomSmartQuery components work in a concurrent environment. Applications which instantiate TCustomSmartQuery descendants may notify each other about their activity on a shared database and be updated each time the database gets modified.

Setting SmartRefresh property to False indicates that concurrent sessions will refresh their datasets on their own.

 Request Support DAC Forum Provide Feedback

5.30.1.1.2.6 SmartState Property

Defines if TCustomSmartQuery is in view mode.

Class

TCustomSmartQuery

Syntax

```
property SmartState: TSmartState;
```

Remarks

Check SmartState property to learn whether TCustomSmartQuery is in view mode.

TCustomSmartQuery is in view mode (SmartState is dsView) after calling View method.

See Also

- View

Request Support DAC Forum Provide Feedback

5.30.1.1.3 Methods

Methods of the **TCustomSmartQuery** class.

For a complete list of the **TCustomSmartQuery** class members, see the TCustomSmartQuery Members topic.

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.

CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that

	matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.

GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.

RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

View	Allows viewing all fields of the updating table.

See Also

- TCustomSmartQuery Class
- TCustomSmartQuery Class Members

 Request Support DAC Forum Provide Feedback

5.30.1.1.3.1 View Method

Allows viewing all fields of the updating table.

Class

TCustomSmartQuery

Syntax

```
procedure View;
```

Remarks

Useful when Expand is True. Lets view all fields of the updating table. Call Cancel method to return dataset to dsBrowse mode. To indicate view mode check SmartState. After calling View method SmartState is dsView.

See Also

- SmartState

 Request Support DAC Forum Provide Feedback

5.30.1.1.4 Events

Events of the **TCustomSmartQuery** class.

For a complete list of the **TCustomSmartQuery** class members, see the TCustomSmartQuery Members topic.

Public

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterSmartRefresh	Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.
RefreshFields	Occures before an application posts changes for the current record to the database or cache.

See Also

- TCustomSmartQuery Class
- TCustomSmartQuery Class Members

5.30.1.1.4.1 AfterSmartRefresh Event

Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.

Class

TCustomSmartQuery

Syntax

```
property AfterSmartRefresh: TDataSetNotifyEvent;
```

Remarks

Occurs every time after TCustomOraQuery performes Smart Refresh procedure.

See Also

- SmartRefresh

5.30.1.1.4.2 RefreshFields Event

Occures before an application posts changes for the current record to the database or cache.

Class

TCustomSmartQuery

Syntax

```
property RefreshFields: TDataSetNotifyEvent;
```

Remarks

Occurs before an application posts changes for the current record to the database or cache.

When Expand is True, write RefreshFields event handler to assign a value to the nonupdating fields.

Example

```
procedure quEmpRefreshFields(DataSet: TDataSet);
begin
```

```
    DataSet.FieldByName('DNAME').AsString:=
      quDept.FieldByName('DNAME').AsString;
end;
```


Request Support DAC Forum Provide Feedback

5.30.1.2 TOraTable Class

A component for retrieving and updating data in a single table without writing SQL statements.

For a list of all members of this type, see TOraTable members.

Unit

OraSmart

Syntax

```
TOraTable = class(TCustomSmartQuery);
```

Remarks

The TOraTable component allows retrieving and updating data in a single table without writing SQL statements. Use TOraTable to access data in a table . Use the TableName property to specify table name. TOraTable uses the KeyFields property to build SQL statements for updating table data. KeyFields is a string containing a semicolon-delimited list of the field names. If KeyFields is not defined before opening, TOraTable uses primary or unique key or ROWID pseudo field.

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet
      TCustomOraQuery
        TCustomSmartQuery
          TOraTable

See Also

- TSmartQuery
- Updating Data with ODAC Dataset Components

- Master/Detail Relationships

5.30.1.2.1 Members

TOraTable class overview.

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and

	types of its parameters.
DependEvents (inherited from **TCustomSmartQuery**)	Used to get or set the names of the events that dataset will depend on in TCustomSmartQuery.SmartRefresh mode.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
Expand (inherited from **TCustomSmartQuery**)	Lets all data controls be aware of all the fields belonging to updating table.
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL

	statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode	Used to specify what kind of lock will be performed when editing a record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the

	master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TCustomSmartQuery**)	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
OrderFields	Used to build ORDER BY clause of SQL statements.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshEvent (inherited from **TCustomSmartQuery**)	Used to get or set the name of the event that dataset will be waiting for in TCustomSmartQuery.SmartRefresh mode.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted,

	updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SmartRefresh (inherited from **TCustomSmartQuery**)	Let TCustomSmartQuery components work in a concurrent environment.
SmartState (inherited from **TCustomSmartQuery**)	Defines if TCustomSmartQuery is in view mode.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the

	TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
TableName	Used to specify the name of the database table this component encapsulates.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.

ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
EmptyTable	Truncates the current table content on the server.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.

Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.

GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.

MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Determines KeyFields and builds query of TOraTable.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range

	of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.
View (inherited from **TCustomSmartQuery**)	Allows viewing all fields of the updating table.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterSmartRefresh (inherited from **TCustomSmartQuery**)	Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.

OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.
RefreshFields (inherited from **TCustomSmartQuery**)	Occures before an application posts changes for the current record to the database or cache.

Request Support DAC Forum Provide Feedback

5.30.1.2.2 Properties

Properties of the **TOraTable** class.

For a complete list of the **TOraTable** class members, see the TOraTable Members topic.

Public

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.

Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DependEvents (inherited from **TCustomSmartQuery**)	Used to get or set the names of the events that dataset will depend on in TCustomSmartQuery.SmartRefresh mode.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
Expand (inherited from **TCustomSmartQuery**)	Lets all data controls be aware of all the fields belonging to updating table.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.

FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing

	detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TCustomSmartQuery**)	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshEvent (inherited from **TCustomSmartQuery**)	Used to get or set the name of the event that dataset will be waiting for in TCustomSmartQuery.SmartRefresh mode.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.

ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.
SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SmartRefresh (inherited from **TCustomSmartQuery**)	Let TCustomSmartQuery components work in a concurrent environment.
SmartState (inherited from **TCustomSmartQuery**)	Defines if TCustomSmartQuery is in view mode.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when

	opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.
StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
FetchAll	Defines whether to request all records of the query from database server when the dataset is being opened.

LockMode	Used to specify what kind of lock will be performed when editing a record.
OrderFields	Used to build ORDER BY clause of SQL statements.
TableName	Used to specify the name of the database table this component encapsulates.

See Also

- TOraTable Class
- TOraTable Class Members

 Request Support DAC Forum Provide Feedback

5.30.1.2.2.1 FetchAll Property

Defines whether to request all records of the query from database server when the dataset is being opened.

Class

TOraTable

Syntax

```
property FetchAll: boolean;
```

Remarks

When set to True, all records of the query are requested from database server when the dataset is being opened. When set to False, records are retrieved when a data-aware component or a program requests it. If a query can return a lot of records, set this property to False if initial response time is important.

When the FetchAll property is False, the first call to TMemDataSet.Locate and TMemDataSet.LocateEx methods may take a lot of time to retrieve additional records to the client side.

 Request Support DAC Forum Provide Feedback

Reserved.

5.30.1.2.2.2 LockMode Property

Used to specify what kind of lock will be performed when editing a record.

Class

TOraTable

Syntax

```
property LockMode: TLockMode default lmLockImmediate;
```

Remarks

Use the LockMode property to define what kind of lock will be performed when editing a record. Locking a record is useful in creating multi-user applications. It prevents modification of a record by several users at the same time.

Locking is performed by the RefreshRecord method.

The default value is lmNone.

To set pessimistic locking use LockMode = lmLockImmediate, TOraDataSet.CheckMode = cmException. To set optimistic locking use LockMode = lmLockDelayed, CheckMode = cmException.

See Also

- TOraStoredProc.LockMode
- TOraQuery.LockMode

Request Support DAC Forum Provide Feedback

5.30.1.2.2.3 OrderFields Property

Used to build ORDER BY clause of SQL statements.

Class

TOraTable

Syntax

```
property OrderFields: string;
```

Remarks

TOraTable uses the OrderFields property to build ORDER BY clause of SQL statements. To set several field names to this property separate them with commas.

TOraTable is reopened when OrderFields is being changed.

See Also

- TOraTable

Request Support DAC Forum Provide Feedback

5.30.1.2.2.4 TableName Property

Used to specify the name of the database table this component encapsulates.

Class

TOraTable

Syntax

```
property TableName: string;
```

Remarks

Use the TableName property to specify the name of the database table this component encapsulates. If TCustomDADataSet.Connection is assigned If Session is set at design time,select a valid table name from the TableName drop-down list in Object Inspector.

See Also

- TOraTable

Request Support DAC Forum Provide Feedback

5.30.1.2.3 Methods

Methods of the **TOraTable** class.

For a complete list of the **TOraTable** class members, see the TOraTable Members topic.

Public

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.

EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
EmptyTable	Truncates the current table content on the server.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.

GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.
GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.

Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
PrepareSQL	Determines KeyFields and builds query of TOraTable.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.

SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.
View (inherited from **TCustomSmartQuery**)	Allows viewing all fields of the updating table.

See Also

- TOraTable Class
- TOraTable Class Members

5.30.1.2.3.1 EmptyTable Method

Truncates the current table content on the server.

Class

TOraTable

Syntax

```
procedure EmptyTable;
```

Remarks

Call the EmptyTable method to truncate the current table content on the server.

Request Support DAC Forum Provide Feedback

5.30.1.2.3.2 PrepareSQL Method

Determines KeyFields and builds query of TOraTable.

Class

TOraTable

Syntax

```
procedure PrepareSQL;
```

Remarks

Call the PrepareSQL method to determine KeyFields and build query of TOraTable.

PrepareSQL is called implicitly when TOraTable is being opened.

Request Support DAC Forum Provide Feedback

5.30.1.3 TSmartQuery Class

A component providing TCustomSmartQuery.Expand fields feature, that lets all data controls be aware of all the fields belonging to updating table an not only those requested in SELECT clause, and TCustomSmartQuery.SmartRefresh feature (in Professional and Developer

editions only).

For a list of all members of this type, see TSmartQuery members.

Unit

OraSmart

Syntax

```
TSmartQuery = class(TCustomSmartQuery);
```

Remarks

TSmartQuery component is a direct descendant of the TOraDataSet class.

TSmartQuery is an alternative to TOraQuery. It provides TCustomSmartQuery.Expand fields feature, that lets all data controls be aware of all the fields belonging to updating table an not only those requested in SELECT clause, and TCustomSmartQuery.SmartRefresh feature (in Professional and Developer editions only).

Inheritance Hierarchy

TMemDataSet
  TCustomDADataSet
    TOraDataSet
      TCustomOraQuery
        TCustomSmartQuery
          TSmartQuery

See Also

- TOraQuery
- TOraTable
- Updating Data with ODAC Dataset Components

Properties

Name	Description
BaseSQL (inherited from **TCustomDADataSet**)	Used to return SQL text without any changes performed by AddWhere, SetOrderBy, and FilterSQL.
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
ChangeNotification (inherited from **TOraDataSet**)	Used to receive database change notification messages to refresh dataset when required.
CheckMode (inherited from **TOraDataSet**)	Used to define the check mode before editing a record.
CommandTimeout (inherited from **TOraDataSet**)	Sets the wait time before a request is sent to the server to terminate the attempt to execute or fetch the current SQL statement.
Conditions (inherited from **TCustomDADataSet**)	Used to add WHERE conditions to a query
Connection (inherited from **TCustomDADataSet**)	Used to specify a connection object to use to connect to a data store.
Cursor (inherited from **TOraDataSet**)	Used to fetch data from the cursor parameter and cursor field in Oracle 8.
DataTypeMap (inherited from **TCustomDADataSet**)	Used to set data type mapping rules
Debug (inherited from **TCustomDADataSet**)	Used to display the statement that is being executed and the values and types of its parameters.
DependEvents (inherited from **TCustomSmartQuery**)	Used to get or set the names of the events that dataset will depend on in TCustomSmartQuery.SmartRefresh mode.
DetailFields (inherited from **TCustomDADataSet**)	Used to specify the fields that correspond to the foreign key fields from

	MasterFields when building master/detail relationship.
Disconnected (inherited from **TCustomDADataSet**)	Used to keep dataset opened after connection is closed.
DMLRefresh (inherited from **TOraDataSet**)	Used to refresh record by RETURNING clause when insert or update is performed.
Encryption (inherited from **TOraDataSet**)	Used to specify encryption options in a dataset.
Expand (inherited from **TCustomSmartQuery**)	Lets all data controls be aware of all the fields belonging to updating table.
FetchAll (inherited from **TOraDataSet**)	Used to request all records of the query from database server when a dataset is being opened.
FetchRows (inherited from **TCustomDADataSet**)	Used to define the number of rows to be transferred across the network at the same time.
FilterSQL (inherited from **TCustomDADataSet**)	Used to change the WHERE clause of SELECT statement and reopen a query.
FinalSQL (inherited from **TCustomDADataSet**)	Used to return SQL text with all changes performed by AddWhere, SetOrderBy, and FilterSQL, and with expanded macros.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
IsPLSQL (inherited from **TOraDataSet**)	Indicates whether a SQL statement is a PL/SQL block.
IsQuery (inherited from **TOraDataSet**)	Indicates whether SQL statement returns rows or not.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
KeyFields (inherited from **TOraDataSet**)	Used to build SQL

	statements for the SQLDelete, SQLInsert, and SQLUpdate properties if they were empty before updating a database.
KeySequence (inherited from **TOraDataSet**)	Used to specify the name of a sequence that will be used to fill in a key field after a new record is inserted or posted to a database.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
LockMode (inherited from **TOraDataSet**)	Used to define when to perform the locking of an editing record.
MacroCount (inherited from **TCustomDADataSet**)	Used to get the number of macros associated with the Macros property.
Macros (inherited from **TCustomDADataSet**)	Makes it possible to change SQL queries easily.
MasterFields (inherited from **TCustomDADataSet**)	Used to specify the names of one or more fields that are used as foreign keys for dataset when establishing detail/master relationship between it and the dataset specified in MasterSource.
MasterSource (inherited from **TCustomDADataSet**)	Used to specify the data source component which binds current dataset to the master one.
NonBlocking (inherited from **TOraDataSet**)	Used to execute a SQL statement and fetch rows by a separate thread.
Options (inherited from **TCustomSmartQuery**)	Used to specify the behaviour of TCustomSmartQuery object.
OptionsDS (inherited from **TOraDataSet**)	Used to specify the behaviour of

	TOraDataSetObject.
ParamCheck (inherited from **TCustomDADataSet**)	Used to specify whether parameters for the Params property are generated automatically after the SQL property was changed.
ParamCount (inherited from **TCustomDADataSet**)	Used to indicate how many parameters are there in the Params property.
Params (inherited from **TOraDataSet**)	Contains the parameters for a query's SQL statement.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
ReadOnly (inherited from **TCustomDADataSet**)	Used to prevent users from updating, inserting, or deleting data in the dataset.
RefreshEvent (inherited from **TCustomSmartQuery**)	Used to get or set the name of the event that dataset will be waiting for in TCustomSmartQuery.SmartRefresh mode.
RefreshMode (inherited from **TOraDataSet**)	Used to specify when to refresh an editing record.
RefreshOptions (inherited from **TCustomDADataSet**)	Used to indicate when the editing record is refreshed.
ReturnParams (inherited from **TOraDataSet**)	Used to return a new fields value to dataset after insert or update.
RowsAffected (inherited from **TCustomDADataSet**)	Used to indicate the number of rows which were inserted, updated, or deleted during the last query operation.
RowsProcessed (inherited from **TOraDataSet**)	Returns the number of rows processed by a query.
SequenceMode (inherited from **TOraDataSet**)	Used to specify the methods used internally to generate a sequenced field.
Session (inherited from **TOraDataSet**)	Used to specify the session in which dataset will be executed.

SmartFetch (inherited from **TOraDataSet**)	The SmartFetch mode is used for fast navigation through a huge amount of records and to minimize memory consumption.
SmartRefresh (inherited from **TCustomSmartQuery**)	Let TCustomSmartQuery components work in a concurrent environment.
SmartState (inherited from **TCustomSmartQuery**)	Defines if TCustomSmartQuery is in view mode.
SQL (inherited from **TCustomDADataSet**)	Used to provide a SQL statement that a query component executes when its Open method is called.
SQLDelete (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying a deletion to a record.
SQLInsert (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that will be used when applying an insertion to a dataset.
SQLLock (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to perform a record lock.
SQLRecCount (inherited from **TCustomDADataSet**)	Used to specify the SQL statement that is used to get the record count when opening a dataset.
SQLRefresh (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used to refresh current record by calling the TCustomDADataSet.RefreshRecord procedure.
SQLType (inherited from **TOraDataSet**)	Used to get the typecode of the SQL statement being processed by Oracle database server.
SQLUpdate (inherited from **TCustomDADataSet**)	Used to specify a SQL statement that will be used when applying an update to a dataset.

StrictUpdate (inherited from **TOraDataSet**)	Used for TOraDataSet to raise the 'Update failed' exception when the number of updated or deleted records are not equal to 1.
UniDirectional (inherited from **TCustomDADataSet**)	Used if an application does not need bidirectional access to records in the result set.
UpdateObject (inherited from **TOraDataSet**)	Used to specify an update object component which provides SQL statements that perform updates of the read-only datasets.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
AddWhere (inherited from **TCustomDADataSet**)	Adds condition to the WHERE clause of SELECT statement in the SQL property.
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
BreakExec (inherited from **TCustomDADataSet**)	Breaks execution of a SQL statement on the server.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.

CreateBlobStream (inherited from **TCustomDADataSet**)	Used to obtain a stream for reading data from or writing data to a BLOB field, specified by the Field parameter.
CreateProcCall (inherited from **TOraDataSet**)	Generates the stored procedure call.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
DeleteWhere (inherited from **TCustomDADataSet**)	Removes WHERE clause from the SQL property and assigns the BaseSQL property.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
ErrorOffset (inherited from **TOraDataSet**)	Returns the parse error offset.
Execute (inherited from **TCustomDADataSet**)	Overloaded. Executes a SQL statement on the server.
Executing (inherited from **TCustomDADataSet**)	Indicates whether SQL statement is still being executed.
Fetched (inherited from **TCustomDADataSet**)	Used to find out whether TCustomDADataSet has fetched all rows.
Fetching (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is still fetching rows.
FetchingAll (inherited from **TCustomDADataSet**)	Used to learn whether TCustomDADataSet is fetching all rows to the end.
FindKey (inherited from **TCustomDADataSet**)	Searches for a record which contains specified field values.
FindMacro (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
FindNearest (inherited from **TCustomDADataSet**)	Moves the cursor to a specific record or to the first record in the dataset that

	matches or is greater than the values specified in the KeyValues parameter.
FindParam (inherited from **TOraDataSet**)	Determines whether a parameter with the specified name exists in a dataset.
GetArray (inherited from **TOraDataSet**)	Retrieves a TOraArray object for a field when only its name is known.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
GetDataType (inherited from **TCustomDADataSet**)	Returns internal field types defined in the MemData and accompanying modules.
GetErrorPos (inherited from **TOraDataSet**)	Returns a row and column of parse error for a SQL statement.
GetFieldObject (inherited from **TCustomDADataSet**)	Returns a multireference shared object from field.
GetFieldPrecision (inherited from **TCustomDADataSet**)	Retrieves the precision of a number field.
GetFieldScale (inherited from **TCustomDADataSet**)	Retrieves the scale of a number field.
GetFile (inherited from **TOraDataSet**)	Retrieves a TOraFile object for a field with known name.
GetInterval (inherited from **TOraDataSet**)	Retrieves a TOraInterval object for a field with known name.
GetKeyFieldNames (inherited from **TCustomDADataSet**)	Provides a list of available key field names.
GetKeyList (inherited from **TOraDataSet**)	Returns the list of table primary key fields.
GetLob (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetLobLocator (inherited from **TOraDataSet**)	Retrieves a TOraLob object for a field with known name.
GetObject (inherited from **TOraDataSet**)	Retrieves a TOraObject object for a field with known name.

GetOrderBy (inherited from **TCustomDADataSet**)	Retrieves an ORDER BY clause from a SQL statement.
GetRef (inherited from **TOraDataSet**)	Retrieves a TOraRef object for a field with known name.
GetTable (inherited from **TOraDataSet**)	Retrieve a TOraNestTable object for a field with known name.
GetTimeStamp (inherited from **TOraDataSet**)	Retrieves a TOraTimeStamp object for a field with known name.
GotoCurrent (inherited from **TCustomDADataSet**)	Sets the current record in this dataset similar to the current record in another dataset.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Lock (inherited from **TCustomDADataSet**)	Locks the current record.
MacroByName (inherited from **TCustomDADataSet**)	Finds a macro with the specified name.
OpenNext (inherited from **TOraDataSet**)	Opens next cursor or rowset in the statement.
ParamByName (inherited from **TOraDataSet**)	Sets or uses parameter information for a specific parameter based on its name.
Prepare (inherited from **TCustomDADataSet**)	Allocates, opens, and parses cursor for a query.
RefreshRecord (inherited from **TCustomDADataSet**)	Actualizes field values for the current record.
RestoreSQL (inherited from **TCustomDADataSet**)	Restores the SQL property modified by AddWhere and SetOrderBy.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.

RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveSQL (inherited from **TCustomDADataSet**)	Saves the SQL property value to BaseSQL.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetOrderBy (inherited from **TCustomDADataSet**)	Builds an ORDER BY clause of a SELECT statement.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
SQLSaved (inherited from **TCustomDADataSet**)	Determines if the SQL property value was saved to the BaseSQL property.
UnLock (inherited from **TCustomDADataSet**)	Releases a record lock.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

View (inherited from **TCustomSmartQuery**)	Allows viewing all fields of the updating table.

Events

Name	Description
AfterExecute (inherited from **TCustomDADataSet**)	Occurs after a component has executed a query to database.
AfterFetch (inherited from **TCustomDADataSet**)	Occurs after dataset finishes fetching data from server.
AfterSmartRefresh (inherited from **TCustomSmartQuery**)	Occurs after the Smart Refresh procedure was performed by TCustomOraQuery.
AfterUpdateExecute (inherited from **TCustomDADataSet**)	Occurs after executing insert, delete, update, lock and refresh operations.
BeforeFetch (inherited from **TCustomDADataSet**)	Occurs before dataset is going to fetch block of records from the server.
BeforeUpdateExecute (inherited from **TCustomDADataSet**)	Occurs before executing insert, delete, update, lock, and refresh operations.
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.
RefreshFields (inherited from **TCustomSmartQuery**)	Occures before an application posts changes for the current record to the database or cache.

5.30.1.4 TSmartQueryOptions Class

This class allows setting up the behaviour of the TSmartQuery class.

For a list of all members of this type, see TSmartQueryOptions members.

Unit

OraSmart

Syntax

```
TSmartQueryOptions = class(TOraDataSetOptions);
```

Inheritance Hierarchy

TDADataSetOptions
  TOraDataSetOptionsDS
    TOraDataSetOptions
      TSmartQueryOptions

Request Support DAC Forum Provide Feedback

5.30.1.4.1 Members

TSmartQueryOptions class overview.

Properties

Name	Description
AutoClose (inherited from **TOraDataSetOptions**)	Used to close OCI cursor after fetching all rows.
AutoPrepare (inherited from **TDADataSetOptions**)	Used to execute automatic TCustomDADataSet.Prepare on the query execution.
CacheCalcFields (inherited from **TDADataSetOptions**)	Used to enable caching of the TField.Calculated and TField.Lookup fields.
CacheLobs (inherited from **TOraDataSetOptions**)	Used to allocate local memory buffer to hold a copy of the Lob content.
CompressBlobMode (inherited from	Used to store values of the BLOB fields in compressed

TDADataSetOptions)	form.
DefaultValues (inherited from **TOraDataSetOptions**)	Used for TOraDataSet to fill the DefaultExpression property of TField objects by appropriate value.
DeferredLobRead (inherited from **TOraDataSetOptions**)	Used to fetch all Oracle 8 Lob values when they are explicitly requested.
DetailDelay (inherited from **TDADataSetOptions**)	Used to get or set a delay in milliseconds before refreshing detail dataset while navigating master dataset.
EnableBCD (inherited from **TOraDataSetOptions**)	Used to enable currency type. Default value of this option is False.
EnableFMTBCD (inherited from **TOraDataSetOptions**)	Used to enable using FMTBCD instead of float for large integer numbers to keep precision.
ExtendedFieldsInfo (inherited from **TOraDataSetOptions**)	Used to perform an additional query to get information about returned fields and the tables they belong to.
FieldsAsString (inherited from **TOraDataSetOptions**)	Used to treat all non-BLOB fields as being of string datatype.
FieldsOrigin (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to fill the Origin property of the TField objects by appropriate value when opening a dataset.
FlatBuffers (inherited from **TDADataSetOptions**)	Used to control how a dataset treats data of the ftString and ftVarBytes fields.
FullRefresh (inherited from **TOraDataSetOptions**)	Used to refresh fields of all tables by the RefreshRecord method.
HideRowId (inherited from **TOraDataSetOptionsDS**)	Used to display the ROWID column.

InsertAllSetFields (inherited from **TDADataSetOptions**)	Used to include all set dataset fields in the generated INSERT statement
KeepPrepared (inherited from **TOraDataSetOptionsDS**)	Used to keep TOraDataSet prepared after closing.
LocalMasterDetail (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to use local filtering to establish master/detail relationship for detail dataset and does not refer to the server.
LongStrings (inherited from **TDADataSetOptions**)	Used to represent string fields with the length that is greater than 255 as TStringField.
MasterFieldsNullable (inherited from **TDADataSetOptions**)	Allows to use NULL values in the fields by which the relation is built, when generating the query for the Detail tables (when this option is enabled, the performance can get worse).
NumberRange (inherited from **TDADataSetOptions**)	Used to set the MaxValue and MinValue properties of TIntegerField and TFloatField to appropriate values.
PrefetchLobSize (inherited from **TOraDataSetOptions**)	Used to retrieve the LOB length and the LOB data beginning during regular fetch.
PrefetchRows (inherited from **TOraDataSetOptions**)	Used to set the number of rows to be prefetched during the execution of a query.
PrepareUpdateSQL (inherited from **TOraDataSetOptions**)	Used to automatically prepare update queries before execution.
ProcNamedParams (inherited from **TOraDataSetOptions**)	Used to specify a notation method of passing parameter values to the stored PL/SQL object.
QueryRecCount (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to

	perform additional query to get the record count for this SELECT, so the RecordCount property reflects the actual number of records.
QuoteNames (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to quote all database object names in autogenerated SQL statements such as update SQL.
RawAsString (inherited from **TOraDataSetOptions**)	Used to treat all RAW fields as being of string datatype.
ReflectChangeNotify (inherited from **TOraDataSetOptions**)	Used for a dataset component to refresh its data when it gets database change notification messages in response to DML or DDL changes on the objects associated with the dataset query.
RemoveOnRefresh (inherited from **TDADataSetOptions**)	Used for a dataset to locally remove a record that can not be found on the server.
RequiredFields (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to set the Required property of the TField objects for the NOT NULL fields.
ReturnParams (inherited from **TDADataSetOptions**)	Used to return the new value of fields to dataset after insert or update.
ScrollableCursor (inherited from **TOraDataSetOptions**)	Used for TOraDataSet to use scrollable server cursor (available since Oracle 9 only) instead of caching data on the client side.
SetFieldsReadOnly (inherited from **TDADataSetOptions**)	Used for a dataset to set the ReadOnly property to True for all fields that do not belong to UpdatingTable or can not be updated.
StatementCache (inherited from **TOraDataSetOptions**)	Used to get a value indicating whether Oracle

	resources assosiated with the current statement will be cached inside a session.
StrictUpdate (inherited from **TDADataSetOptions**)	Used for TCustomDADataSet to raise an exception when the number of updated or deleted records is not equal 1.
TemporaryLobUpdate (inherited from **TOraDataSetOptions**)	Temporary LOBs are used to write input and input/ output LOB parameters into database when executing dataset's SQL statements.
TrimFixedChar (inherited from **TDADataSetOptions**)	Specifies whether to discard all trailing spaces in the string fields of a dataset.
UpdateAllFields (inherited from **TDADataSetOptions**)	Used to include all dataset fields in the generated UPDATE and INSERT statements.
UpdateBatchSize (inherited from **TDADataSetOptions**)	Used to get or set a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.

Request Support DAC Forum Provide Feedback

5.30.2 Enumerations

Enumerations in the **OraSmart** unit.

Enumerations

Name	Description
TSmartState	Specifies if TCustomSmartQuery is in view mode.

Request Support DAC Forum Provide Feedback

5.30.2.1 TSmartState Enumeration

Specifies if TCustomSmartQuery is in view mode.

Unit

OraSmart

Syntax

```
TSmartState = (dsView);
```

Values

Value	Meaning
dsView	TCustomSmartQuery is in view mode.

Remarks

Check the SmartState property to learn whether TCustomSmartQuery is in view mode.

TCustomSmartQuery is in view mode (SmartState is dsView) after calling View method.

5.31 OraSQLMonitor

This unit contains implementation of the TOraSQLMonitor component.

Classes

Name	Description
TOraSQLMonitor	This component serves for monitoring dynamic SQL execution in ODAC-based applications.

5.31.1 Classes

Classes in the **OraSQLMonitor** unit.

Classes

Name	Description
TOraSQLMonitor	This component serves for monitoring dynamic SQL execution in ODAC-based applications.

Request Support DAC Forum Provide Feedback

5.31.1.1 TOraSQLMonitor Class

This component serves for monitoring dynamic SQL execution in ODAC-based applications.

For a list of all members of this type, see TOraSQLMonitor members.

Unit

OraSQLMonitor

Syntax

```
TOraSQLMonitor = class(TCustomDASQLMonitor);
```

Remarks

Use TOraSQLMonitor to monitor dynamic SQL execution in ODAC-based applications. TOraSQLMonitor provides two ways of displaying debug information: with dialog window, DBMonitor or Borland SQL Monitor. Furthermore to receive debug information the TCustomDASQLMonitor.OnSQL event can be used. Also it is possible to use all these ways at the same time, though an application may have only one TOraSQLMonitor object. If an application has no TOraSQLMonitor instance, the Debug window is available to display SQL statements to be sent.

Inheritance Hierarchy

TCustomDASQLMonitor

TOraSQLMonitor

See Also

- TCustomDADataSet.Debug
- TCustomDASQL.Debug
- DBMonitor

 Request Support DAC Forum Provide Feedback

5.31.1.1.1 Members

TOraSQLMonitor class overview.

Properties

Name	Description
Active (inherited from **TCustomDASQLMonitor**)	Used to activate monitoring of SQL.
DBMonitorOptions (inherited from **TCustomDASQLMonitor**)	Used to set options for dbMonitor.
Options (inherited from **TCustomDASQLMonitor**)	Used to include the desired properties for TCustomDASQLMonitor.
TraceFlags (inherited from **TCustomDASQLMonitor**)	Used to specify which database operations the monitor should track in an application at runtime.

Events

Name	Description
OnSQL (inherited from **TCustomDASQLMonitor**)	Occurs when tracing of SQL activity on database components is needed.

 Request Support DAC Forum Provide Feedback

5.32 OraTransaction

This unit contains implementation of the TOraTransaction component.

Classes

Name	Description
TOraTransaction	A component for managing transactions in an application.

Enumerations

Name	Description
TGlobalCoordinator	Specifies with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.

 Request Support DAC Forum Provide Feedback

5.32.1 Classes

Classes in the **OraTransaction** unit.

Classes

Name	Description
TOraTransaction	A component for managing transactions in an application.

 Request Support DAC Forum Provide Feedback

5.32.1.1 TOraTransaction Class

A component for managing transactions in an application.

For a list of all members of this type, see TOraTransaction members.

Unit

OraTransaction

Syntax

```
TOraTransaction = class(TDATransaction);
```

Remarks

The TOraTransaction component is used to provide discrete transaction control over connection. It can be used for manipulating simple local and global transactions.

Inheritance Hierarchy

TDATransaction

TOraTransaction

See Also

- TOraTransaction Component
- TCustomDAConnection.StartTransaction
- TCustomDAConnection.Commit
- TCustomDAConnection.Rollback
- TOraTransaction Component

5.32.1.1.1 Members

TOraTransaction class overview.

Properties

Name	Description
Active	Used to indicate whether the transaction is active or not.
BranchQualifiers	Used to represent branch qualifier part of XID for transaction branches.

DefaultCloseAction (inherited from **TDATransaction**)	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
DefaultSession	Used to specify the session for performing the transaction.
GlobalCoordinator	Used to determine with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.
InactiveTimeOut	Used to specify the waiting time before deleting inactive transaction branch.
IsolationLevel	Used to specify how the transactions containing database modifications are handled.
ResumeTimeOut	Used to set the wait time to resume the transaction branch if it is used by another session.
Sessions	Used to specify a session for the given index.
SessionsCount	Used to provide the number of sessions associated with the transaction component.
TransactionId	Used to represent global transaction identifier.
TransactionName	Used to assign a name to the current transaction.

Methods

Name	Description
AddSession	Overloaded. Associates a TOraSession component with the transaction component.
ClearSessions	Disassociates the transaction component from

	all its session components.
Commit (inherited from **TDATransaction**)	Commits the current transaction.
Detach	Deactivates a transaction.
RemoveSession	Disassociates the specified session from the transaction.
Resume	Resumes a detached transaction.
Rollback (inherited from **TDATransaction**)	Discards all modifications of data associated with the current transaction and ends the transaction.
RollbackToSavepoint	Discards all modifications made during the current transaction and restores its state to the moment of the savepoint.
Savepoint	Defines a savepoint in the transaction.
StartTransaction	Overloaded. Begins a new user transaction against the database server.

Events

Name	Description
OnCommit (inherited from **TDATransaction**)	Occurs after the transaction has been successfully committed.
OnCommitRetaining (inherited from **TDATransaction**)	Occurs after CommitRetaining has been executed.
OnError	Occurs for processing errors that can be arised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.
OnRollback (inherited from **TDATransaction**)	Occurs after the transaction has been successfully rolled back.

OnRollbackRetaining (inherited from **TDATransaction**)	Occurs after RollbackRetaining has been executed.

Request Support DAC Forum Provide Feedback

5.32.1.1.2 Properties

Properties of the **TOraTransaction** class.

For a complete list of the **TOraTransaction** class members, see the TOraTransaction Members topic.

Public

Name	Description
Active	Used to indicate whether the transaction is active or not.
BranchQualifiers	Used to represent branch qualifier part of XID for transaction branches.
DefaultCloseAction (inherited from **TDATransaction**)	Used to specify the transaction behaviour when it is destroyed while being active, or when one of its connections is closed with the active transaction.
Sessions	Used to specify a session for the given index.
SessionsCount	Used to provide the number of sessions associated with the transaction component.
TransactionId	Used to represent global transaction identifier.

Published

Name	Description
DefaultSession	Used to specify the session for performing the transaction.

GlobalCoordinator	Used to determine with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.
InactiveTimeOut	Used to specify the waiting time before deleting inactive transaction branch.
IsolationLevel	Used to specify how the transactions containing database modifications are handled.
ResumeTimeOut	Used to set the wait time to resume the transaction branch if it is used by another session.
TransactionName	Used to assign a name to the current transaction.

See Also

- TOraTransaction Class
- TOraTransaction Class Members

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.1 Active Property

Used to indicate whether the transaction is active or not.

Class

TOraTransaction

Syntax

```
property Active: boolean;
```

Remarks

Use the Active property to indicate whether the transaction is active or not.

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.2 BranchQualifiers Property(Indexer)

Used to represent branch qualifier part of XID for transaction branches.

Class

TOraTransaction

Syntax

```
property BranchQualifiers[Index: integer]: TBytes;
```

Parameters

Index

Holds a branch qualifier index.

Remarks

Use the BranchQualifiers property to represent a branch qualifier part of XID for transaction branches.

Request Support DAC Forum Provide Feedback

5.32.1.1.2.3 DefaultSession Property

Used to specify the session for performing the transaction.

Class

TOraTransaction

Syntax

```
property DefaultSession: TOraSession;
```

Remarks

Use the DefaultSession property to specify the session which is used to perform the transaction. For distributed transactions use the TOraTransaction.AddSession method instead.

See Also

- TOraTransaction.AddSession

5.32.1.1.2.4 GlobalCoordinator Property

Used to determine with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.

Class

TOraTransaction

Syntax

```
property GlobalCoordinator: TGlobalCoordinator default
gcInternal;
```

Remarks

Use the GlobalCoordinator property to determine with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.

5.32.1.1.2.5 InactiveTimeOut Property

Used to specify the waiting time before deleting inactive transaction branch.

Class

TOraTransaction

Syntax

```
property InactiveTimeOut: integer default 0;
```

Remarks

Use the InactiveTimeOut property to set for server the time to wait before deleting inactive

transaction branch.

Request Support DAC Forum Provide Feedback

5.32.1.1.2.6 IsolationLevel Property

Used to specify how the transactions containing database modifications are handled.

Class

TOraTransaction

Syntax

```
property IsolationLevel: TOraIsolationLevel default
ilReadCommitted;
```

Remarks

Use the IsolationLevel property to specify how the transactions containing database modifications are handled.

Request Support DAC Forum Provide Feedback

5.32.1.1.2.7 ResumeTimeOut Property

Used to set the wait time to resume the transaction branch if it is used by another session.

Class

TOraTransaction

Syntax

```
property ResumeTimeOut: integer default 0;
```

Remarks

Use the ResumeTimeOut property for setting the wait time to resume the transaction branch if it is used by another session.

See Also

- Resume

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.8 Sessions Property(Indexer)

Used to specify a session for the given index.

Class

TOraTransaction

Syntax

```
property Sessions[Index: integer]: TOraSession;
```

Parameters

Index

Holds the index to specify a session for.

Remarks

Use the Sessions property to specify a session for the given index.

See Also

- SessionsCount
- RemoveSession
- TOraTransaction.AddSession

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.9 SessionsCount Property

Used to provide the number of sessions associated with the transaction component.

Class

TOraTransaction

Syntax

```
property SessionsCount: integer;
```

Remarks

Use the SessionsCount property to get the number of sessions associated with the transaction component.

See Also

- Sessions
- RemoveSession
- TOraTransaction.AddSession

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.10 TransactionId Property

Used to represent global transaction identifier.

Class

TOraTransaction

Syntax

```
property TransactionId: TBytes;
```

Remarks

Use the TransactionId property to represent global transaction identifier which is the part of XID. Server associates it with local transaction.

 Request Support DAC Forum Provide Feedback

5.32.1.1.2.11 TransactionName Property

Used to assign a name to the current transaction.

Class

TOraTransaction

Syntax

```
property TransactionName: string;
```

Remarks

Use the TransactionName property to assign a name to the current transaction. This parameter is useful in distributed database environments when you have to identify and resolve in-doubt transactions. The text string is limited to 255 bytes.

Request Support DAC Forum Provide Feedback

5.32.1.1.3 Methods

Methods of the **TOraTransaction** class.

For a complete list of the **TOraTransaction** class members, see the TOraTransaction Members topic.

Public

Name	Description
AddSession	Overloaded. Associates a TOraSession component with the transaction component.
ClearSessions	Disassociates the transaction component from all its session components.
Commit (inherited from **TDATransaction**)	Commits the current transaction.
Detach	Deactivates a transaction.
RemoveSession	Disassociates the specified session from the transaction.
Resume	Resumes a detached transaction.
Rollback (inherited from **TDATransaction**)	Discards all modifications of data associated with the current transaction and ends the transaction.
RollbackToSavepoint	Discards all modifications

	made during the current transaction and restores its state to the moment of the savepoint.
Savepoint	Defines a savepoint in the transaction.
StartTransaction	Overloaded. Begins a new user transaction against the database server.

See Also

- TOraTransaction Class
- TOraTransaction Class Members

Request Support DAC Forum Provide Feedback

5.32.1.1.3.1 AddSession Method

Associates a TOraSession component with the transaction component.

Class

TOraTransaction

Overload List

Name	Description
AddSession(Session: TOraSession)	Associates a TOraSession component with the transaction component.
AddSession(Session: TOraSession; BranchQualifier: TBytes)	Associates a TOraSession component with the transaction component.

Request Support DAC Forum Provide Feedback

Associates a TOraSession component with the transaction component.

Class

TOraTransaction

Syntax

```
procedure AddSession(Session: TOraSession); overload;
```

Parameters

Session
Holds a TOraSession component

Remarks

Call the AddSession method to associate a TOraSession component with the transaction component.

Request Support DAC Forum Provide Feedback

Associates a TOraSession component with the transaction component.

Class

TOraTransaction

Syntax

```
procedure AddSession(Session: TOraSession; BranchQualifier:
TBytes); overload;
```

Parameters

Session
Holds a TOraSession component

BranchQualifier
Holds a branch qualifier.

See Also

- TOraTransaction.Sessions
- TOraTransaction.RemoveSession
- TOraTransaction.ClearSessions

Request Support DAC Forum Provide Feedback

5.32.1.1.3.2 ClearSessions Method

Disassociates the transaction component from all its session components.

Class

TOraTransaction

Syntax

```
procedure ClearSessions;
```

Remarks

Call the ClearSessions method to disassociate the transaction component from all its session components.

See Also

- Sessions
- AddSession
- RemoveSession

 Request Support DAC Forum Provide Feedback

5.32.1.1.3.3 Detach Method

Deactivates a transaction.

Class

TOraTransaction

Syntax

```
procedure Detach;
```

Remarks

Call the Detach method to make a transaction inactive.

See Also

- Resume

 Request Support DAC Forum Provide Feedback

5.32.1.1.3.4 RemoveSession Method

Disassociates the specified session from the transaction.

Class

TOraTransaction

Syntax

```
procedure RemoveSession(Session: TOraSession);
```

Parameters

Session
Holds a TOraSession object.

Remarks

Call the RemoveSession method to disassociate the specified session from the transaction.

See Also

- Sessions
- TOraTransaction.AddSession
- ClearSessions

 Request Support DAC Forum Provide Feedback

5.32.1.1.3.5 Resume Method

Resumes a detached transaction.

Class

TOraTransaction

Syntax

```
procedure Resume;
```

Remarks

Call the Resume method to resume a detached transaction.

See Also

- ResumeTimeOut
- Detach

 Request Support DAC Forum Provide Feedback

5.32.1.1.3.6 RollbackToSavepoint Method

Discards all modifications made during the current transaction and restores its state to the moment of the savepoint.

Class

TOraTransaction

Syntax

```
procedure RollbackToSavepoint(const Savepoint: string);
```

Parameters

Savepoint

Holds the name of the savepoint.

Remarks

Call the RollbackToSavepoint method to cancel all updates for the current transaction and restore its state up to the moment of the last defined savepoint.

See Also

- Savepoint
- TDATransaction.Rollback

 Request Support DAC Forum Provide Feedback

5.32.1.1.3.7 Savepoint Method

Defines a savepoint in the transaction.

Class

TOraTransaction

Syntax

```
procedure Savepoint(const Savepoint: string);
```

Parameters

Savepoint

Holds the savepoint name that identifies the savepoint.

Remarks

Call the Savepoint method to define a point in the transaction to which you can roll back later. As the parameter, you can pass any valid name to identify the savepoint.

To roll back to the last savepoint call RollbackToSavepoint.

See Also

- RollbackToSavepoint

5.32.1.1.3.8 StartTransaction Method

Begins a new user transaction against the database server.

Class

TOraTransaction

Overload List

Name	Description
StartTransaction	Begins a new user transaction against the database server.
StartTransaction(Resume: boolean)	Begins a new user transaction against the database server.

 Request Support DAC Forum Provide Feedback

Begins a new user transaction against the database server.

Class

TOraTransaction

Syntax

```
procedure StartTransaction; overload; override;
```

© 1997-2024 Devart. All Rights Reserved. Request Support DAC Forum Provide Feedback

Begins a new user transaction against the database server.

Class

TOraTransaction

Syntax

```
procedure StartTransaction(Resume: boolean); reintroduce;
overload;
```

Parameters

Resume

True, if detached transaction branches will be resumed on sessions. False otherwise.

Remarks

Call the StartTransaction method to begin a new user transaction against the database server. Before calling StartTransaction, an application should check the status of the TOraTransaction.Active property. If TOraTransaction.Active is True, it indicates that a transaction is already in progress, a subsequent call to StartTransaction without first calling TDATransaction.Commit or TDATransaction.Rollback to end the current transaction raises EDatabaseError. Calling StartTransaction when connection is closed also raises EDatabaseError.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by

the server until an application calls Commit to save the changes or Rollback to cancel them.

Setting the Resume parameter to True means detached transaction branches will be resumed on sessions.

 Request Support DAC Forum Provide Feedback

5.32.1.1.4 Events

Events of the **TOraTransaction** class.

For a complete list of the **TOraTransaction** class members, see the TOraTransaction Members topic.

Public

Name	Description
OnCommit (inherited from **TDATransaction**)	Occurs after the transaction has been successfully committed.
OnCommitRetaining (inherited from **TDATransaction**)	Occurs after CommitRetaining has been executed.
OnRollback (inherited from **TDATransaction**)	Occurs after the transaction has been successfully rolled back.
OnRollbackRetaining (inherited from **TDATransaction**)	Occurs after RollbackRetaining has been executed.

Published

Name	Description
OnError	Occurs for processing errors that can be arised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.

See Also

- TOraTransaction Class
- TOraTransaction Class Members

 Request Support DAC Forum Provide Feedback

5.32.1.1.4.1 OnError Event

Occurs for processing errors that can be arised during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others.

Class

TOraTransaction

Syntax

```
property OnError: TDATransactionErrorEvent;
```

Remarks

Write the OnError event handler to process errors that occur during executing transaction and savepoint control statements such as COMMIT, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT and others. Check the E parameter to get an error code.

 Request Support DAC Forum Provide Feedback

5.32.2 Enumerations

Enumerations in the **OraTransaction** unit.

Enumerations

Name	Description
TGlobalCoordinator	Specifies with what distributed transaction, perform two-phase commit or rollback on all sessions

	will be coordinated.

 Request Support DAC Forum Provide Feedback

5.32.2.1 TGlobalCoordinator Enumeration

Specifies with what distributed transaction, perform two-phase commit or rollback on all sessions will be coordinated.

Unit

OraTransaction

Syntax

```
TGlobalCoordinator = (gcInternal, gcMTS);
```

Values

Value	Meaning
gcInternal	Transaction will be coordinated by the transaction component internally.
gcMTS	Transaction will be coordinated by Microsoft Transaction Server DTC.

 Request Support DAC Forum Provide Feedback

5.33 VirtualDataSet

This unit contains implementation of the TVirtualDataSet component.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Request Support DAC Forum Provide Feedback

5.33.1 Classes

Classes in the **VirtualDataSet** unit.

Classes

Name	Description
TCustomVirtualDataSet	A base class for representation of arbitrary data in tabular form.
TVirtualDataSet	Dataset that processes arbitrary non-tabular data.

Request Support DAC Forum Provide Feedback

5.33.1.1 TCustomVirtualDataSet Class

A base class for representation of arbitrary data in tabular form.

For a list of all members of this type, see TCustomVirtualDataSet members.

Unit

VirtualDataSet

Syntax

```
TCustomVirtualDataSet = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet

TCustomVirtualDataSet

5.33.1.1.1 Members

TCustomVirtualDataSet class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.

Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.

LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are

	enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.33.1.2 TVirtualDataSet Class

Dataset that processes arbitrary non-tabular data.

For a list of all members of this type, see TVirtualDataSet members.

Unit

VirtualDataSet

Syntax

```
TVirtualDataSet = class(TCustomVirtualDataSet);
```

Inheritance Hierarchy

TMemDataSet
  TCustomVirtualDataSet
    TVirtualDataSet

Request Support DAC Forum Provide Feedback

5.33.1.2.1 Members

TVirtualDataSet class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.

CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending

	values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

5.33.2 Types

Types in the **VirtualDataSet** unit.

Types

Name	Description
TOnDeleteRecordEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.
TOnGetFieldValueEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.
TOnGetRecordCountEvent	This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.
TOnModifyRecordEvent	This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Request Support DAC Forum Provide Feedback

5.33.2.1 TOnDeleteRecordEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnDeleteRecord event.

Unit

VirtualDataSet

Syntax

```
TOnDeleteRecordEvent = procedure (Sender: TObject; RecNo:
Integer) of object;
```

Parameters

Sender

An object that raised the event.

RecNo

Number of the record being deleted.

 Request Support DAC Forum Provide Feedback

5.33.2.2 TOnGetFieldValueEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetFieldValue event.

Unit

VirtualDataSet

Syntax

```
TOnGetFieldValueEvent = procedure (Sender: TObject; Field: TField;
RecNo: Integer; out Value: Variant) of object;
```

Parameters

Sender

An object that raised the event.

Field

The field, which data has to be returned.

RecNo

The number of the record, which data has to be returned.

Value

Requested field value.

 Request Support DAC Forum Provide Feedback

5.33.2.3 TOnGetRecordCountEvent Procedure Reference

This type is used for the E:Devart.Dac.TVirtualDataSet.OnGetRecordCount event.

Unit

VirtualDataSet

Syntax

```
TOnGetRecordCountEvent = procedure (Sender: TObject; out Count:
Integer) of object;
```

Parameters

Sender
An object that raised the event.
Count
The number of records that the virtual dataset will contain.
 Request Support DAC Forum Provide Feedback

5.33.2.4 TOnModifyRecordEvent Procedure Reference

This type is used for E:Devart.Dac.TVirtualDataSet.OnInsertRecord and E:Devart.Dac.TVirtualDataSet.OnModifyRecord events.

Unit

VirtualDataSet

Syntax

```
TOnModifyRecordEvent = procedure (Sender: TObject; var RecNo:
Integer) of object;
```

Parameters

Sender
An object that raised the event.
RecNo
Number of the record being inserted or modified.
 Request Support DAC Forum Provide Feedback

5.34 VirtualTable

This unit contains implementation of the TVirtualTable component.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

 Request Support DAC Forum Provide Feedback

5.34.1 Classes

Classes in the **VirtualTable** unit.

Classes

Name	Description
TVirtualTable	Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

 Request Support DAC Forum Provide Feedback

5.34.1.1 TVirtualTable Class

Dataset that stores data in memory. This component is placed on the Data Access page of the Component palette.

For a list of all members of this type, see TVirtualTable members.

Unit

VirtualTable

Syntax

```
TVirtualTable = class(TMemDataSet);
```

Inheritance Hierarchy

TMemDataSet

TVirtualTable

 Request Support DAC Forum Provide Feedback

5.34.1.1.1 Members

TVirtualTable class overview.

Properties

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
DefaultSortType	Used to determine the default type of local sorting for string fields.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Methods

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the

	dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
Assign	Copies fields and data from another TDataSet component.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
LoadFromFile	Loads data from a file into a TVirtualTable component.
LoadFromStream	Copies data from a stream into a TVirtualTable component.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.

Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

Events

Name	Description
OnUpdateError (inherited from **TMemDataSet**)	Occurs when an exception is generated while cached updates are applied to a database.
OnUpdateRecord (inherited from **TMemDataSet**)	Occurs when a single update component can not handle the updates.

Request Support DAC Forum Provide Feedback

5.34.1.1.2 Properties

Properties of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the TVirtualTable Members topic.

Public

Name	Description
CachedUpdates (inherited from **TMemDataSet**)	Used to enable or disable the use of cached updates for a dataset.
IndexFieldNames (inherited from **TMemDataSet**)	Used to get or set the list of fields on which the recordset is sorted.
KeyExclusive (inherited from **TMemDataSet**)	Specifies the upper and lower boundaries for a range.
LocalConstraints (inherited from **TMemDataSet**)	Used to avoid setting the Required property of a TField component for NOT NULL fields at the time of opening TMemDataSet.
LocalUpdate (inherited from **TMemDataSet**)	Used to prevent implicit update of rows on database server.
Prepared (inherited from **TMemDataSet**)	Determines whether a query is prepared for execution or not.
Ranged (inherited from **TMemDataSet**)	Indicates whether a range is

	applied to a dataset.
UpdateRecordTypes (inherited from **TMemDataSet**)	Used to indicate the update status for the current record when cached updates are enabled.
UpdatesPending (inherited from **TMemDataSet**)	Used to check the status of the cached updates buffer.

Published

Name	Description
DefaultSortType	Used to determine the default type of local sorting for string fields.

See Also

- TVirtualTable Class
- TVirtualTable Class Members

 Request Support DAC Forum Provide Feedback

5.34.1.1.2.1 DefaultSortType Property

Used to determine the default type of local sorting for string fields.

Class

TVirtualTable

Syntax

```
property DefaultSortType: TSortType default stCaseSensitive;
```

Remarks

The DefaultSortType property is used when a sort type is not specified explicitly after the field name in the TMemDataSet.IndexFieldNames property of a dataset.

 Request Support DAC Forum Provide Feedback

5.34.1.1.3 Methods

Methods of the **TVirtualTable** class.

For a complete list of the **TVirtualTable** class members, see the TVirtualTable Members topic.

Public

Name	Description
ApplyRange (inherited from **TMemDataSet**)	Applies a range to the dataset.
ApplyUpdates (inherited from **TMemDataSet**)	Overloaded. Writes dataset's pending cached updates to a database.
Assign	Copies fields and data from another TDataSet component.
CancelRange (inherited from **TMemDataSet**)	Removes any ranges currently in effect for a dataset.
CancelUpdates (inherited from **TMemDataSet**)	Clears all pending cached updates from cache and restores dataset in its prior state.
CommitUpdates (inherited from **TMemDataSet**)	Clears the cached updates buffer.
DeferredPost (inherited from **TMemDataSet**)	Makes permanent changes to the database server.
EditRangeEnd (inherited from **TMemDataSet**)	Enables changing the ending value for an existing range.
EditRangeStart (inherited from **TMemDataSet**)	Enables changing the starting value for an existing range.
GetBlob (inherited from **TMemDataSet**)	Overloaded. Retrieves TBlob object for a field or current record when only its name or the field itself is known.
LoadFromFile	Loads data from a file into a TVirtualTable component.
LoadFromStream	Copies data from a stream into a TVirtualTable

	component.
Locate (inherited from **TMemDataSet**)	Overloaded. Searches a dataset for a specific record and positions the cursor on it.
LocateEx (inherited from **TMemDataSet**)	Overloaded. Excludes features that don't need to be included to the TMemDataSet.Locate method of TDataSet.
Prepare (inherited from **TMemDataSet**)	Allocates resources and creates field components for a dataset.
RestoreUpdates (inherited from **TMemDataSet**)	Marks all records in the cache of updates as unapplied.
RevertRecord (inherited from **TMemDataSet**)	Cancels changes made to the current record when cached updates are enabled.
SaveToXML (inherited from **TMemDataSet**)	Overloaded. Saves the current dataset data to a file or a stream in the XML format compatible with ADO format.
SetRange (inherited from **TMemDataSet**)	Sets the starting and ending values of a range, and applies it.
SetRangeEnd (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the end of the range of rows to include in the dataset.
SetRangeStart (inherited from **TMemDataSet**)	Indicates that subsequent assignments to field values specify the start of the range of rows to include in the dataset.
UnPrepare (inherited from **TMemDataSet**)	Frees the resources allocated for a previously prepared query on the server and client sides.
UpdateResult (inherited from **TMemDataSet**)	Reads the status of the latest call to the ApplyUpdates method while

	cached updates are enabled.
UpdateStatus (inherited from **TMemDataSet**)	Indicates the current update status for the dataset when cached updates are enabled.

See Also

- TVirtualTable Class
- TVirtualTable Class Members

5.34.1.1.3.1 Assign Method

Copies fields and data from another TDataSet component.

Class

TVirtualTable

Syntax

```
procedure Assign(Source: TPersistent); override;
```

Parameters

Source

Holds the TDataSet component to copy fields and data from.

Remarks

Call the Assign method to copy fields and data from another TDataSet component.

Note: Unsupported field types are skipped (i.e. destination dataset will contain less fields than the source one). This may happen when Source is not a TVirtualTable component but some server-oriented dataset.

Example

```
Query1.SQL.Text := 'SELECT * FROM DEPT';
Query1.Active := True;
VirtualTable1.Assign(Query1);
VirtualTable1.Active := True;
```

```
OraQuery1.SQL.Text := 'SELECT * FROM DEPT';
OraQuery1.Active := True;
VirtualTable1.Assign(OraQuery1);
VirtualTable1.Active := True;
```

See Also

- TVirtualTable

Request Support DAC Forum Provide Feedback

5.34.1.1.3.2 LoadFromFile Method

Loads data from a file into a TVirtualTable component.

Class

TVirtualTable

Syntax

```
procedure LoadFromFile(const FileName: string; LoadFields:
boolean = True; DecodeHTMLEntities: boolean = True);
```

Parameters

FileName
Holds the name of the file to load data from.

LoadFields
Indicates whether to load fields from the file.

DecodeHTMLEntities
Indicates whether to decode HTML entities from the file.

Remarks

Call the LoadFromFile method to load data from a file into a TVirtualTable component. Specify the name of the file to load into the field as the value of the FileName parameter.This file may be an XML document in ADO-compatible format or in virtual table data format. The file format is detected automatically.

Request Support DAC Forum Provide Feedback

5.34.1.1.3.3 LoadFromStream Method

Copies data from a stream into a TVirtualTable component.

Class

TVirtualTable

Syntax

```
procedure LoadFromStream(Stream: TStream; LoadFields: boolean =
True; DecodeHTMLEntities: boolean = True);
```

Parameters

Stream
Holds the stream from which the field's value is copied.

LoadFields
Indicates whether to load fields from the stream.

DecodeHTMLEntities
Indicates whether to decode HTML entities from the stream.

Remarks

Call the LoadFromStream method to copy data from a stream into a TVirtualTable component. Specify the stream from which the field's value is copied as the value of the Stream parameter. Data in the stream may be in ADO-compatible format or in virtual table data format. The data format is detected automatically.